МАГІСТЕРСЬКА РОБОТА

МР.ІПм – 19.00.00.000 **ПЗ**

Група ІПм-22-5

Алексієнков Владислав

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Алексієнков Владислав Костянтинович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та засоби контролю автоматичних оновлень в системах IoT

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Алексієнков В.К.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Піх Марія Михайлівна, асистент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

В.о. завідувача кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

_____

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
В.о. зав. кафедрою **ІПЗ**

доц. В.В. Бандура

“ 04 ” вересня 2023 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Алексієнкову Владиславу Костянтиновичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “Моделі, методи та засоби контролю автоматичних оновлень в системах IoT”

керівник проекту (роботи) Піх Марія Михайлівна, асистент

затверджені наказом закладу вищої освіти від “ 18 ” грудня 2023 р. № 738/7

2. Строк подання студентом проекту (роботи) 25 січня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних та програмних технологій IoT

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

1. Теоретичні основи побудови автоматичних оновлень
2. Дослідження методів захисту інформації та методології побудови систем Інтернету речей
3. Реалізація інформаційної технології контролю автоматичних оновлень в системах IoT
4. Оцінка отриманих результатів

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

1. Огляд Інтернету речей (рис. 1.1)
2. Склад RFID-мітки (рис. 1.3.)
3. Пропонована загальна архітектура мережі (рис. 2.2)
4. Вікно підказок для різних сценаріїв (рис. 1.4)
5. База даних програми на сервері Tomcat (рис. 3.2)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Нормоконтроль	доц., к.т.н. Вовк Р.Б.	
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2023 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	01.10.2023	виконано
2	Теоретичні основи побудови автоматичних оновлень	25.10.2023	виконано
3	Дослідження методів захисту інформації та методології побудови систем інтернету речей	10.11.2023	виконано
4	Реалізація інформаційної технології контролю автоматичних оновлень в системах IoT	01.12.2023	виконано
5	Оцінка отриманих результатів	23.12.2023	виконано
6	Затвердження пояснювальної записки роботи завідувачем кафедри	25.01.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 77 с., 37 рис., 4 табл., 42 джерела.

Тема: Моделі, методи та засоби контролю автоматичних оновлень в системах IoT

Об'єкт дослідження: системи Інтернету речей (IoT)

Мета роботи: є розробка та вдосконалення моделей, методів та засобів контролю автоматичних оновлень в системах Інтернету речей (IoT) з метою підвищення безпеки та ефективності функціонування цих систем.

Предмет дослідження: моделі контролю автоматичних оновлень в системах IoT, методи реалізації автоматичних оновлень, засоби та технології, що використовуються для забезпечення безпеки та ефективності автоматичних оновлень.

Результати дослідження:

Очікується, що дане дослідження призведе до розробки нових, більш ефективних моделей та методів контролю автоматичних оновлень для систем Інтернету речей. Результатом буде покращення безпеки, цілісності та ефективності оновлень, що сприятиме стабільному та безпечному функціонуванню систем IoT.

Висновок:

Отже, в рамках цього дослідження було досягнуто поставленої мети — розроблено та вдосконалено моделі, методи та засоби контролю автоматичних оновлень для підвищення безпеки та ефективності систем Інтернету речей. Запропоновано модифікований метод захисту інформації в мережі Інтернету речей, який базується на REST API та на методі еліптично-кривої криптографії.

ЦИФРОВИЙ ПІДПИС, ІНТЕРНЕТ РЕЧЕЙ, АДМІНІСТРАТОР, ДИЗАЙН РІШЕННЯ, МЕНЕДЖЕР, ЗАВАНТАЖУВАЧ, АЛГОРИТМ, ЗАХИСТ ІНФОРМАЦІЇ

ANNOTATION

Master's thesis: 77 pages, 37 figures, 4 tables, 42 sources.

Topic: Models, methods and means of controlling automatic updates in IoT systems

Research object: Internet of Things (IoT) systems

The purpose of the work: is the development and improvement of models, methods and means of controlling automatic updates in Internet of Things (IoT) systems in order to improve the security and efficiency of the functioning of these systems.

Subject of research: automatic update control models in IoT systems, methods of implementing automatic updates, tools and technologies used to ensure the safety and efficiency of automatic updates.

Research results:

It is expected that this research will lead to the development of new, more efficient models and methods of automatic update control for the Internet of Things system. The result will be improved security, integrity and efficiency of updates, which will contribute to the stable and secure functioning of the IoT system.

Conclusion:

So, within the framework of this study, the set goal was achieved — to develop and improve models, methods and means of automatic update control to improve the security and efficiency of the Internet of Things system. A modified method of protecting information in the Internet of Things network is proposed, which is based on the REST API and the method of elliptic-curve cryptography.

DIGITAL SIGNATURE, INTERNET OF THINGS, ADMINISTRATOR, SOLUTION DESIGN, MANAGER, LOADER, ALGORITHM, INFORMATION PROTECTION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ASF - Apache Software Foundation

DSA - Digital Signature Algorithm

DSU - Dynamic software update

IOT - Internet of Things

ITU - International Telecommunications Union

JAR - Java Archive

JKS - Java KeyStore

JNLP - Java network launching protocol

JRE - Java Runtime Environment

JWS - Java Web Start

MD5 - Message-Digest Algorithm

MIT - Massachusetts Institute of Technology

RFID - Radio-Frequency Identification

SHA - Secure Hash Algorithm

SSL - Secure Sockets Layer

SWT - Standard Widget Toolkit

TLS - Transport Layer Security

ВСТУП

Актуальність роботи

Забезпечення інформаційної безпеки є однією з найбільш актуальних проблем в галузі інформаційних технологій на сьогодні. У сучасному світі актуалізуються рішення «Інтернету речей» (Internet of things, IoT), завдання забезпечення інформаційної безпеки в яких також має досить актуальний та основоположний характер.

Сьогодні загальноприйняте визначення «Інтернету речей» наступне: динамічна глобальна мережева інфраструктура з самостійним налаштуванням можливостей на основі стандартних та сумісних протоколів зв'язку, де фізичні та віртуальні «речі» мають ідентифікатори, фізичні атрибути та віртуальні персоналії, використовують інтелектуальні інтерфейси [4]. Проєкт з впровадження промислового «Інтернету речей» вже було реалізовано у таких галузях, як сільське господарство, харчова промисловість, екологічний моніторинг, відеоспостереження та ін. Для того, щоб забезпечити оптимальне впровадження IoT-пристроїв у промислових умовах. Галузеву специфіку та вимоги до таких факторів, як вартість, безпека, конфіденційність та ризики, необхідно усвідомити ще до того, як «Інтернет речей» почне широко використовуватись у промисловості.

Основною проблемою, що виникає при використанні мереж IoT, є відсутність захисту від несанкціонованого впливу. У найкращому разі, така атака з боку зловмисника може стати причиною завдання шкоди майну людини, а в найгіршому заподіяти шкоду здоров'ю людини. Пристрої можуть бути скомпрометовані зловмисником, які мають вихід до Інтернету. Виробляючи контроль над такими пристроями, хакер може здійснити відключення будь-якого електричного обладнання, у тому числі й тих, які є своєрідними системами життєзабезпечення, системами охорони на виробництві.

Актуальність даної роботи визначається кількістю та швидкістю розвитку Інтернету речей (IoT) і, відповідно, зростанням числа підключених пристроїв. Оновлення програмного та апаратного забезпечення цих пристроїв стає критично

важливим завданням для забезпечення їхньої безпеки, ефективності та функціональності. Актуальність дослідження полягає в необхідності розробки нових моделей та методів для забезпечення безпечного та ефективного процесу автоматичного оновлення в системах IoT.

Безпека IoT - з поглибленням використання IoT-пристроїв у побуті, промисловості та інших сферах життя зростає ймовірність кібератак. Ефективний контроль автоматичних оновлень стає ключовим фактором для запобігання вразливостям та забезпечення безпеки.

Оптимізація ресурсів - збільшення кількості підключених пристроїв також ставить завдання оптимізації ресурсів. Вдосконалені методи контролю оновлень дозволяють ефективніше використовувати обмежені ресурси IoT-пристроїв.

Сумісність та інтероперабельність - оновлення також є важливим для забезпечення сумісності різних пристроїв та підтримки інтероперабельності, особливо в умовах швидкого розвитку стандартів та технологій.

Порівняння роботи з відомими розв’язаннями проблеми

Порівняння роботи з існуючими розв'язаннями дозволяє визначити конкурентні переваги та новизну запропонованих методів. Порівняно з традиційними підходами, розроблені моделі можуть більш ефективно враховувати особливості розподілених та обмежених за ресурсами пристроїв IoT.

Вдосконалені методи забезпечення безпеки та шифрування дозволяють уникнути ризиків, пов'язаних із зловживанням оновлень та забезпечити конфіденційність даних.

В порівнянні зі стандартними рішеннями, розроблені методи можуть дозволити більш ефективне використання обмежених ресурсів, що є важливим для пристроїв IoT. Порівняння дозволяє визначити переваги запропонованого підходу та виправлення можливих недоліків, підсилюючи важливість результатів дослідження.

Мета і задачі дослідження

Метою магістерської роботи є розробка та вдосконалення моделей, методів та засобів контролю автоматичних оновлень в системах Інтернету речей (IoT) з метою підвищення безпеки та ефективності функціонування цих систем.

Досягнення мети включало розв’язання таких **задач**:

- Аналіз існуючих моделей та методів

- Розробка нових моделей контролю оновлень в системах Інтернету речей

- Оцінка безпеки в системах Інтернету речей

- Дослідження методів захисту інформації та методології побудови систем Інтернету речей;

Реалізація інформаційної технології контролю автоматичних оновлень в системах IoT.

Об’єктом дослідження є системи Інтернету речей (IoT).

IoT представляє собою мережу підключених пристроїв, які обмінюються даними та взаємодіють між собою без прямого втручання людини.

Предметом дослідження є моделі, методи та засоби контролю автоматичних оновлень в системах Інтернету речей (IoT). Дослідження спрямоване на вивчення та вдосконалення процесу автоматичного оновлення програмного та апаратного забезпечення в розподілених середовищах IoT..

Методи дослідження

Для досягнення поставленої мети будуть використанні такі методи: літературний огляд, моделювання, системний аналіз, теорія порівняння, експериментальний аналіз, програмна імплементація.

Наукова новизна отриманих результатів

Наукова новизна полягає в адаптації контролю оновлень до умов IoT та в розробці методів, що забезпечують безпеку та ефективність оновлень в цьому конкретному контексті. Адаптація до специфіки IoT, забезпечення цілісності та конфіденційності, експериментальне тестування та імплементація:

Практичне значення одержаних результатів

Отримані результати мають велике значення для підвищення безпеки IoT-систем: Розроблені моделі та методи контролю автоматичних оновлень сприятимуть покращенню безпеки в системах Інтернету речей. Це дозволить уникати потенційних загроз та забезпечить стабільну роботу підключених пристроїв. Засоби, розроблені в рамках дослідження, сприятимуть оптимізації процесу оновлень,

зменшенню споживання ресурсів та підвищенню ефективності систем IoT. Це може мати позитивний вплив на економічну сторону розробки та утримання IoT-проектів. Інтеграція в реальні проекти: Результати дослідження можуть бути використані розробниками та архітекторами IoT-проектів для імплементації нових засобів контролю оновлень. Це розширить можливості підтримки безпеки та підвищить ефективність реальних систем.

Особистий внесок студента

В рамках цього дослідження було досягнуто поставленої мети — розроблено та вдосконалено моделі, методи та засоби контролю автоматичних оновлень для підвищення безпеки та ефективності систем Інтернету речей. Результати можуть бути використані в реальних IoT-проектах для забезпечення стабільного та безпечного функціонування підключених пристроїв.

Структура та обсяг магістерської роботи

Магістерська робота викладена на 77 сторінках друкованого тексту, який складається із вступу, трьох розділів, висновків, списку використаних джерел (42 найменування). Робота містить 37 рисунків і 4 таблиці.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПОБУДОВИ АВТОМАТИЧНИХ ОНОВЛЕНЬ В СИСТЕМАХ ІНТЕРНЕТ РЕЧЕЙ

1.1 Основні відомості про концепцію Інтернет речей

Ми стали свідками розквіту ери IoT (Інтернет речей). Коротко кажучи, «Інтернет речей» означає таку мережу, де повсякденні об’єкти з’єднуються з Інтернетом [4]. IoT зробив Інтернет повсюдним у повсякденному житті людини, і це повсюдне поширення IoT не лише принесло людині зручність, але й сприяло розвитку багатьох сфер, таких як індустрія мікросхем, бездротовий зв’язок, безпека, розподілена система, хмарні обчислення та так далі. Сенс Інтернету речей полягає в тому, що він забезпечує безперебійний зв’язок між фізичним простором і цифровим простором. Певною мірою застосування IoT принесло революцію в життя людини. Як показано на рис. 1.1, Інтернет речей приносить користь людям у багатьох аспектах.

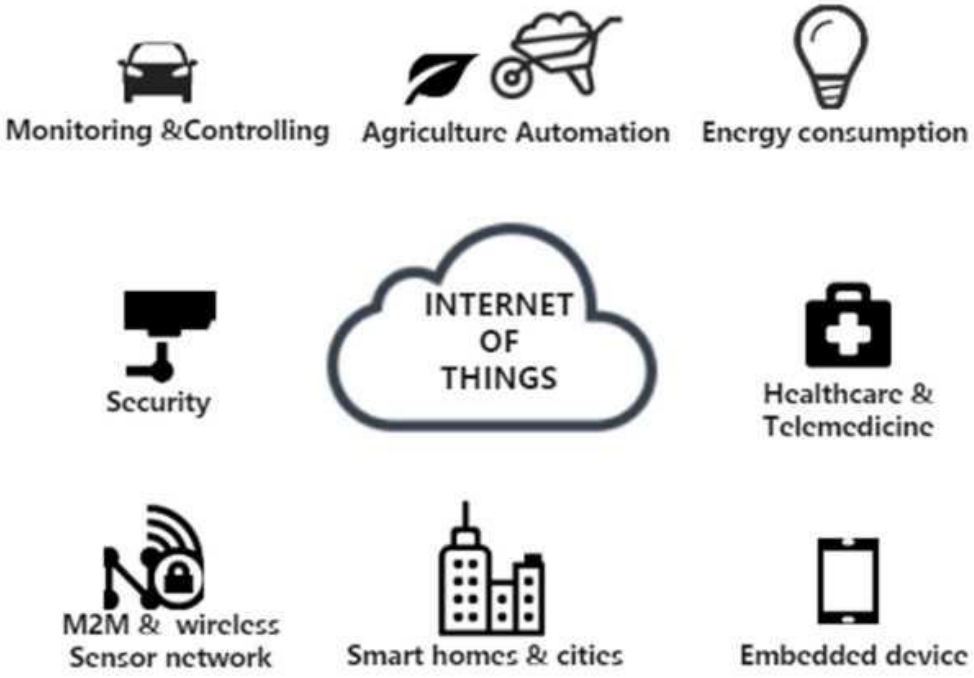

Рис. 1.1. Огляд Інтернету речей

Концепція Інтернету речей була вперше офіційно запропонована Кевіном Ештоном, Девідом Броком і Санджаєм Шармою у 1999 році [5], Auto-ID Lab є всесвітньою дослідницькою організацією для сфера Інтернету речей. Тоді тему Інтернету речей знову озвучив ITU (Міжнародний союз телекомунікацій), і цього разу Інтернет речей вийшов на інший рівень. Після цього дослідження Інтернету речей різко зросли, особливо в останні роки, відразу з'явилися різноманітні інтелектуальні програми.

Термін «Інтернет речей» складається з двох термінів, один — «Інтернет», інший — «речі» [2]. Існує два значення Інтернету речей відповідно до двох термінів. По-перше, ядром і базовою частиною IOT все ще є «Інтернет». По-друге, користувачі поширюються на будь-які «Речі», які можуть потрапити в обмін інформацією та спілкування. Існує багато визначень Інтернету речей. Строго кажучи, це визначається як: з'єднує елементи з Інтернетом для обміну ресурсами та зв'язку відповідно до певних протоколів за допомогою обладнання сприйняття інформації (такого як радіочастотна ідентифікація) і для досягнення певної мети [6].

Термін «радіочастотна ідентифікація (RFID)» був запропонований Кевіном Ештоном, який є засновником Auto-ID Lab [7]. RFID зчитує інформацію, що зберігається в мітці, за допомогою радіохвиль. Перевага пристрою RFID порівняно зі штрих-кодом полягає в тому, що він вимагає прямої видимості для відстеження зчитувача [8].

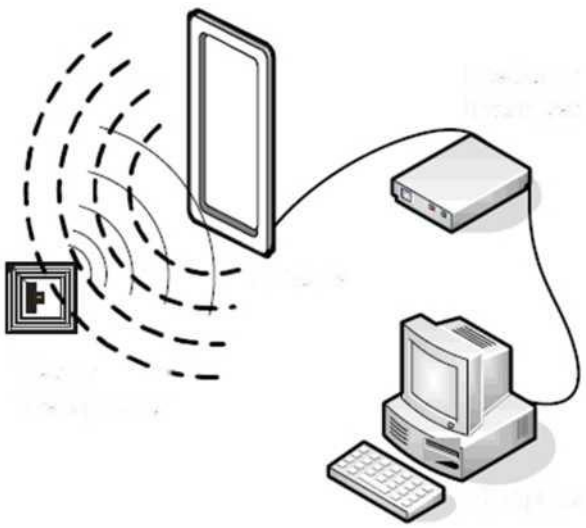

Рис. 1.2. Склад RFID (радіочастотна ідентифікація)

Система RFID складається з двох частин. Один — тег, інший — зчитувач, зображений на рис. 1.2. Тег використовується для збору інформації, а зчитувач використовується для передачі інформації на комп'ютер. Після виконання цієї роботи комп’ютер обробить інформацію, а потім виконає певну поведінку.

Як показано на рис. 1.3, RFID-мітка відноситься до пристрою, який оснащений антеною та мікророзривником і використовує RFID-мітку для відстеження об’єктів [9]. Частина мітки також має дві частини: одна антена для прийому та передачі сигналу, один мікрочіп як накопичувач для зберігання та обробки інформації. Тег зазначений унікальними номерами для ідентифікації.

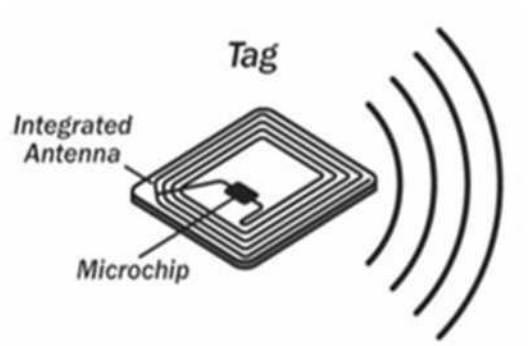

Рис. 1.3. Склад RFID-мітки [9]

Незважаючи на те, що шлях Інтернету речей проходить дуже швидко, все ще існує багато проблем, які перешкоджають його розвитку. Наприклад, питання безпеки, потужність, складність фреймворку IoT, підключення, масштабованість тощо. Отже, для подолання цих викликів у майбутньому потрібно зробити багато речей.

Захищений зв’язок стосується двох об’єктів, які спілкуються один з одним безпечним способом. Чим довше буде спілкування підвищити ймовірність бути скомпрометованими. З цієї причини пропонується багато безпечної теорії та технології шифрування. У цьому розділі буде представлено кілька основних алгоритмів шифрування, які будуть застосовані в цьому проекті.

Алгоритм безпечного хешування скорочено називається SHA, який є одним із сімейства криптографічних хеш-функцій [10]. Його можна застосовувати для

алгоритму цифрового підпису (DSA). Насправді дайджест повідомлення — це рядок фіксованої довжини, який називається «хеш-значення». Наприклад, для повідомлення з розміром менше 2 в степені 64 біти, SHA1 як сімейство SHA створить 160-бітний дайджест повідомлення, як показано на рис. 1.4.

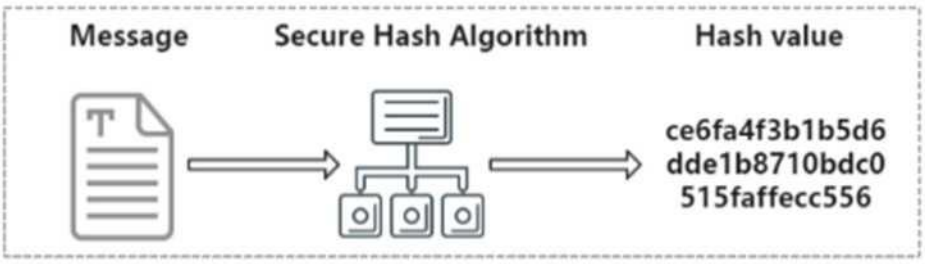

Рис. 1.4. Базовий процес генерації дайджесту повідомлення

Дайджест повідомлення дуже важливий, оскільки він використовуватиметься для перевірки цілісності даних, коли повідомлення буде отримано одержувачем. Дайджест повідомлення зберігається в певному місці повідомлення або файлу та передається з носієм, таким як повідомлення або файл. Якщо повідомлення пошкоджено, одержувач створить інший дайджест повідомлення під час його повторного обчислення.

Незворотність: вихідне повідомлення не можна відновити з дайджесту повідомлення. Поки що SHA все ще вважається досить безпечним, на відміну від алгоритму MD5 Message-Digest, який було зламано.

Унікальність: два різних повідомлення ніколи не створять того самого дайджесту повідомлення. Це гіпотеза з великою ймовірністю, тому що ймовірність появи того самого дайджесту повідомлення вкрай мала, але зазвичай вона ігнорується.

1.2 Використання цифрового підпису в IoT

Підпис у традиційному розумінні відноситься до почерку, який описує чиєсь ім'я або інший знак, який зазвичай пишуть на документах як доказ особи. Крім того, крім їх різних проявів. Цифровий підпис використовується для демонстрації

автентичності цифрового документа. Дійсний цифровий підпис повинен гарантувати три безпечні елементи повідомлення: цілісність, конфіденційність, неспростовність.

Цілісність: відсутність несанкціонованої зміни інформації. Іншими словами, гарантуйте, що джерело, адресат повідомлення правильні, а вміст повідомлення неможливо підробити.

Конфіденційність: відсутність несанкціонованого розголошення інформації. Іншими словами, гарантуйте, що повідомлення не буде витік неавторизованою організацією.

Невідмовність: гарантія того, що відправник і одержувач повідомлення не можуть заперечувати, що вони виконали операцію.

Цифровий підпис є одним із застосувань алгоритмів асиметричного шифрування, також відомого як криптографія з відкритим ключем. Технологія цифрового підпису зазвичай відноситься до трьох алгоритмів, алгоритму генерації ключів, алгоритму підпису та алгоритму перевірки, які виражаються (G, S, V) [11].

Алгоритм генерації ключів. Генератор ключів (G), який використовує алгоритм відкритого ключа, такий як знаменитий RSA, для генерації пар ключів: закритого ключа prk і відповідного відкритого ключа pk.

Алгоритм підписання. Signing(S) підписує повідомлення закритим ключем для створення підпису. Введіть рядок s і закритий ключ prk, повертає підпис ss.

Алгоритм перевірки. Перевірка (V) перевіряє підпис відкритим ключем, а потім підтверджує автентичність повідомлення відповідно до різних результатів. Введіть рядок s, підпис ss і відкритий ключ pk і виведіть прийняти або відхилити.

Етап підписання цифрового підпису зображено на рис. 1.5. Після генерації відкритого та закритого ключів інструмент підписання створить одностороннє хеш-значення вихідних даних із хеш-функцією. Потім зашифруйте хеш-значення за допомогою закритого ключа, щоб створити зашифрований хеш, який називається підписом. Сертифікат використовуватиметься для розшифровки підпису на етапі перевірки.

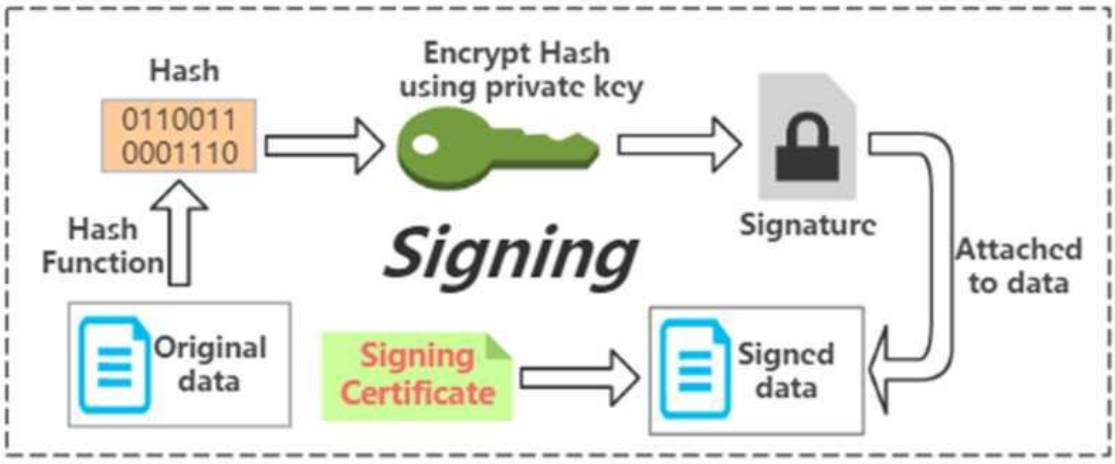

Рис. 1.5. Процес підписання цифрового підпису

Етап перевірки цифрового підпису показано на рис. 1.6. Хеш-значення є унікальним і необоротним. Це означає, що будь-яка зміна даних, навіть одного символу, призведе до величезних змін. Ця функція допомагає одержувачу перевірити цілісність даних шляхом розшифровки цифрового підпису за допомогою відкритого ключа відправника, яким є сертифікат, і отримати хеш-значення h. У той же час хеш-значення вихідних даних, отриманих із підписаних даних, обчислюється знову, припускаючи, що хеш-значення дорівнює h'. Якщо h дорівнює хеш-значенню h', то судіть, що дані не змінені. Якщо h не дорівнює хеш-значенню h', то дані вважаються такими, що хтось втрутився, тому цілісність даних порушується.

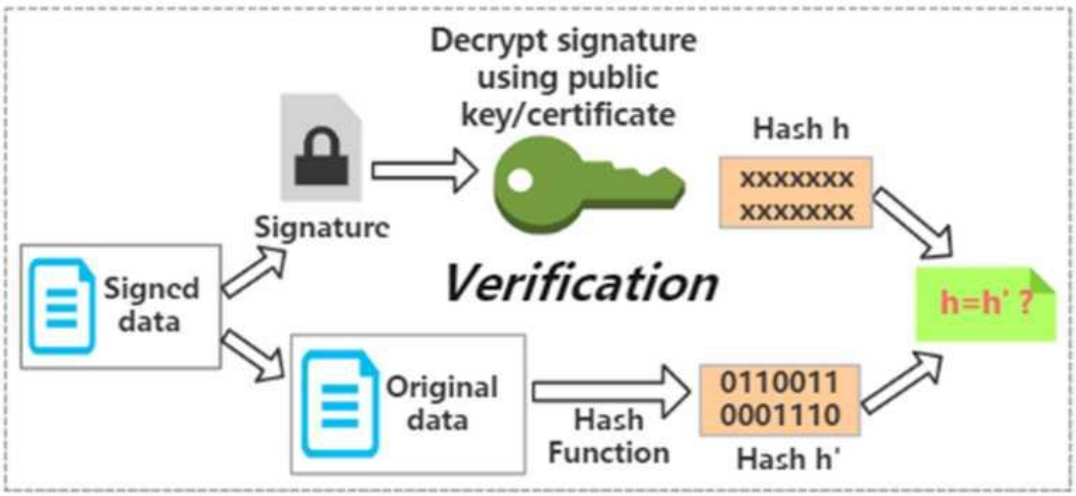

Рис. 1.6. Процес перевірки цифрового підпису

Крім того, він також може підтвердити, чи збігається приватний ключ з відповідним відкритим ключем, тому автентифікація може бути гарантованою. Крім того, дуже важко спростувати підписання документа, оскільки власник конкретного

Застосування цифрового підпису надзвичайно широке. Зараз у багатьох країнах цифрові підписи електронних документів розглядаються як юридичні підписи, і відповідні департаменти Сполучених Штатів опублікували деякі закони та правила цифрового підпису.

1.3 Рівень захищених сокетів

Система зв'язку є важливою частиною Інтернету, і питання безпеки зв'язку завжди є важливим питанням. Разом зі створенням сучасної системи зв'язку люди знайшли кілька можливих і ефективних способів гарантувати безпеку цифрового зв'язку за допомогою певної математичної теорії. Основна концепція полягає в шифруванні вмісту спілкування між двома сторонами на випадок, якщо вміст буде легко перехоплено перехоплювачами.

Рівень захищених сокетів, скорочено SSL, є одним із способів вирішення вищезазначених проблем. SSL – це різновид протоколу безпеки, призначеного для забезпечення безпеки Інтернет-зв'язку. Вперше він був запропонований у 1994 році компанією Netscape, а потім широко використовується в цифрових комунікаціях. На даний момент SSL і наступна версія TLS (Transport Layer Security) є відносно розвиненим протоколом безпечного зв'язку і розроблений для інтерфейсу, щоб полегшити використання програмістами кількох мов програмування, включаючи Java. Вони часто служать у режимі клієнт-сервер для встановлення зашифрованого каналу зв'язку.

З точки зору функціональних аспектів, протокол SSL/TLS можна розділити на два підпротоколи: протокол запису SSL/TLS і протокол рукостискання [12], як показано на рис. 1.7. Протокол Record заснований на протоколі TCP/IP, який забезпечує надійний канал передачі даних і деякі інші основні функції. Протокол Handshake покладається на свій нижчий протокол Record, який використовується

для автентифікації один з одним, узгодження відповідного алгоритму та обміну ключами сеансу перед фактичним початком передачі даних.

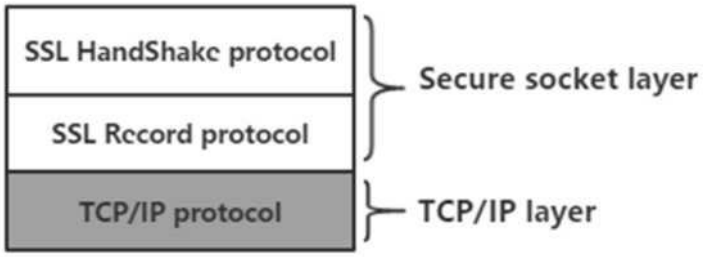

Рис. 1.7. Два підпротоколи протоколу захищеного сокета

З точки зору напрямку сертифікації, зв'язок SSL/TLS також можна розділити на два типи: односторонню автентифікацію SSL/TLS і двосторонню автентифікацію [13]. Рис. 1.8 ілюструє ці два типи способів автентифікації.

Рис. 1.8. Процес автентифікації зв'язку SSL.

Одностороння автентифікація SSL/TLS: існує лише одностороння сертифікація між двома сторонами. Спочатку клієнт ініціює запит рукостискання. По-друге, після того, як сервер отримає його, сервер вибере відповідну версію протоколу та метод шифрування для цього конкретного клієнта. Потім сервер надішле цьому клієнту результат узгодження та його сертифікат. По-третє, клієнт надішле на сервер ключ

сеансу, зашифрований відкритим ключем (сертифікатом) сервера. По-четверте, сервер розшифрує зашифровані дані за допомогою власного закритого ключа. Тепер ці дві сторони можуть почати безпечний зв’язок SSL за допомогою ключа сеансу, який відомий лише цим двом сторонам, тому наступні передані дані будуть зашифровані за допомогою ключа сеансу.

Двостороння автентифікація SSL/TLS: обидві сторони будуть взаємною автентифікацією. Іншими словами, обидві сторони обміняються відкритими ключами (сертифікатами). Цей базовий процес, імовірно, передбачає односторонню автентифікацію SSL/TLS, лише невелика різниця полягає в кількох додаткових кроках у фазі узгодження, як показано в пунктирній рамці на рис. 1.8. Після того, як сервер надішле клієнту результат узгодження та сертифікат, він запитає сертифікат клієнта. А потім клієнт надсилає свій сертифікат на сервер. Крім того, клієнт надсилає на сервер свій цифровий підпис, створений за допомогою власного закритого ключа, а сервер перевіряє легітимність цифрового підпису за допомогою сертифіката клієнта (відкритого ключа). Подальші кроки такі ж, як і одностороння автентифікація. Нарешті, ці дві сторони встановлюють захищений зв’язок через сокет і можуть розпочати безпечний обмін повідомленнями.

1.4 Автоматичне оновлення програмного забезпечення в системах Інтернету речей

Зараз багато операційних систем і програм забезпечують функцію автоматичного оновлення, щоб покращити роботу користувача або з інших причин. Потрібно опанувати деякі необхідні базові знання, пов’язані з автоматичним оновленням.

Динамічне оновлення програмного забезпечення скорочено називається DSU. DSU відноситься до області досліджень, що програма оновлення повинна виконуватися під час її роботи. У минулому не було багато досліджень і застосувань DSU, оскільки кількість пристроїв невелика, навіть мала. Однак зараз усе по-

іншому, Інтернет речей принесе величезну кількість пристроїв, тому DSU показує ширший простір у міру розвитку Інтернету речей.

Мета DSU — зробити процес оновлення безперервним. Процес динамічного оновлення можна розглядати як такий, що включає три частини відповідно до посилання [14], а саме динамічне зв'язування, повторне зв'язування та передача стану відповідно.

Динамічне зв'язування: Ресурс нової версії має бути доступним для конкретної програми під час її роботи. Точніше, інструмент, призначений для завантаження даних, спочатку отримує доступ до ресурсів із спільних бібліотек, а потім поміщає їх у певне місце, наприклад, адресний простір. Після цього програма може знаходити ці ресурси з бібліотеки лише за їх іменами, оскільки подальша робота передається певному інструменту чи механізму.

Повторне зв'язування: після виконання процесу динамічного зв'язування не виникне проблем із завантаженням нових ресурсів, таких як код, оскільки програму буде пов'язано з правильним місцем. Але для старих ресурсів це не те саме, тому потрібен процес повторного зв'язування. Процес повторного прив'язування призведе до того, що наявний сліпий файл у старій версії повторно прив'яжеться до нового місця.

Передача стану: загалом структуру даних, швидше за все, потрібно буде змінити після завершення процесу оновлення програми. Наприклад, одним типом структури даних є ціла форма в старій версії, але яка змінена на рядкову форму в новій версії. Інший приклад — розширення сховища. База даних програми може збільшитися після оновлення до нової версії.

Певною мірою автоматичне оновлення програмного забезпечення є розширенням динамічних оновлень програмного забезпечення, але різниця полягає в тому, що автоматичне оновлення потребує завершення програми поточної версії під час виконання процесу оновлення, тоді як динамічне оновлення цього робити не потрібно. Принципи більшості програм або систем автоматичного оновлення схожі. Основний принцип роботи можна розділити на дві частини відповідно до основної функції: завантажувач і оновлення.

Завантажувач: по-перше, завантажувач перевіряє нову версію програми, яка зазвичай регулярно з'являється як файл jar у базі даних. База даних може бути веб-сайтом (URL) або іншим сховищем форм. По-друге, завантажувач перевірить файл на наявність проблем із безпекою. Форма перевірки буде змінюватись залежно від різних вимог програми або системи. По-третє, після проходження перевірки завантажувач завантажить файл програми нової версії та збереже його у визначеному місці. Більше того, він сповістить програму оновлення для оновлення.

Оновлення: по-перше, програма оновлення чекатиме повідомлення від завантажувача. Якщо він отримає сповіщення від завантажувача, він перевірить своєчасність нової завантаженої програми. Якщо номер поточної версії менший за номер завантаженої програми, програма перейде до наступного кроку. По-друге, програма оновлення визначить, чи запущено поточну версію програми. Якщо він запущений, програма оновлення завершить запущену програму та зробить резервну копію для програми поточної версії, якщо процес оновлення не вдасться. По-третє, програма оновлення виконає процес встановлення нової версії програми. Протягом відповідно до вимог користувача. Навіть нічого не відображати, щоб користувач відчував себе прозорим. По-четверте, програма оновлення перезапустить пристрій, коли завершиться процес інсталяції, і запустить пристрій із новою версією програми під час запуску пристрою.

Коротше кажучи, завантажувач відповідає за завантаження нової версії програми або системи, а програма оновлення відповідає за її оновлення.

Щоб краще запропонувати рішення та реалізувати цей проект, нам потрібно вивчити кілька існуючих пов'язаних робіт. Ці існуючі успішні роботи можуть бути використані як частина вирішення постановки проблеми цього проекту. Спочатку ми вивчимо рішення Windows Update, потім рішення Firefox Update і, нарешті, рішення Java Web Start.

Традиційні операційні системи зазвичай містять певне рішення для автоматичного оновлення. Наприклад, у повсякденному житті ми можемо часто стикатися з оновленнями системи, як-от з Windows 7 на Windows 10 або з IOS 10.0 на IOS 10.3.1 Windows Update є одним із найвідоміших програм для автоматичного

оновлення, але спочатку Windows Update не підтримує автоматичне оновлення до випуску ME у 2000 році.

Windows Update включає автоматичне завантаження та встановлення. Клієнт автоматичного оновлення щодня перевірятиме елементи оновлення на сервері Windows Update. Тоді користувач може завантажити доступні елементи оновлення та негайно встановити або просто повідомити користувача про завантаження цих елементів заздалегідь.

Точніше, коли користувач вмикає функцію Windows Update, комп'ютер користувача підключається до сервера веб-сайту. Деякі спеціальні інструменти сканuватимуть операційні системи користувача та перевірятимуть, чи існують деякі елементи, які потрібно оновити. Якщо існує, користувачеві буде запропоновано завантажити цей файл оновлення. Після завантаження цих файлів і встановлення завершено, це ще не кінець, користувач повинен перезавантажити свій комп'ютер, щоб оновити середовище, щоб завершити це оновлення. Для інсталяції оновлення користувачеві потрібно натиснути кнопку, яка вказує на функцію інсталяції, а потім інсталювати її. Але насправді файл Windows Update неможливо виконати, оскільки імена цих файлів закінчуються форматом .MSU, а не .exe [15]. Цей тип суфікса призначений для Windows Update.

Java Web Start, скорочено JWS, є програмною технологією, іншими словами, це програма. Отримати Java Web Start легко, якщо на комп'ютері користувача встановлено Java 5.0 або пізнішу версію, оскільки JWS є плагіном Java Runtime Environment (JRE). Це означає, що Java Web Start буде автоматично встановлено на комп'ютері після встановлення Java.

Java Web Start змушує користувача завантажувати відповідні файли програми з веб-сайту та запускає програму в локальній системі користувача. Програма Java Web Start має кілька переваг. По-перше, він надає зручний спосіб активації та запуску програми «Один раз». По-друге, JWS може гарантувати, що система завжди запускатиме останню версію програми. По-третє, він простий у використанні, уникає складного процесу встановлення та оновлення [16]. Є три різні способи

запуску програми: з веб-браузера, робочого столу та Java Cache Viewer. Цей звіт буде зосереджений на підході до запуску веб-браузера.

Java Web Start вимагає від розробника налаштування файлу JNLP (протоколу запуску мережі Java), цей файл вважається тригером [17]. Файл JNLP ініціює процес завантаження для певної програми, яка часто відображається як файли Jar або Ear. Крім того, файл JNLP можна розглядати як міст між Java Web Start і веб-браузером (сервером), який спрямовує Java Web Start до місця завантаження файлів програми. Для безпеки ці файли мають бути підписані, інакше Java Web Start відмовиться завантажувати непідписані файли. Програма Java Web Start регулярно перевірятиме наявність файлу новішої версії. Якщо є, він завантажить ці файли та замінить старіші, які зберігаються в кеш-пам'яті локальної системи, щоб ці файли завжди були актуальними. Більше того, інструмент під назвою jardiff допомагає розробнику створювати різні та збільшувати версії додатків, на основі цієї методики він може гарантувати, що програма Java Web Start одночасно отримує лише одну версію програми.

Хоча програма Java Web Start є хорошим інструментом для автоматичного оновлення, є деякі обмеження. Один полягає в тому, що для розповсюдження цих файлів потрібен веб-сервер, наприклад сервер Tomcat. Іншим обмеженням є те, що для цього потрібне підключення до Інтернету, що іноді знижує його доступність тощо.

Getdown [18] — це програма з відкритим вихідним кодом, призначена для автоматичного завантаження та встановлення для системи, яку можна завантажити на веб-сайті Github. Ця програма розроблена ТОВ

(Three Rings Design), яка є компанією з розробки онлайн-ігор. Основна мета Getdown — розповсюджувати та підтримувати всі файли, пов'язані з програмою. Як згадувалося в попередньому розділі, Java Web Start має певні обмеження, тому Getdown створено для компенсації цих обмежень і як заміну JWS.

Програма Getdown складається з кількох важливих керуючих файлів, таких як getdown.txt і digest.txt. Файл getdown повідомляє системі про завантажену URL-адресу та робочий каталог, яку версію файлу Jar слід завантажити та деяку іншу

основну інформацію про цільовий файл Jar. Файл дайджесту розроблено з метою безпеки. Іноді Інтернет може не працювати, тому завантажені файли не завершуються, які пошкоджені. Запуск цих пошкоджених файлів непередбачуваний. Отже, необхідно створити файл дайджесту. Дайджест.txt містить дайджест кожного файлу, який можна використовувати для перевірки легітимності цільових файлів.

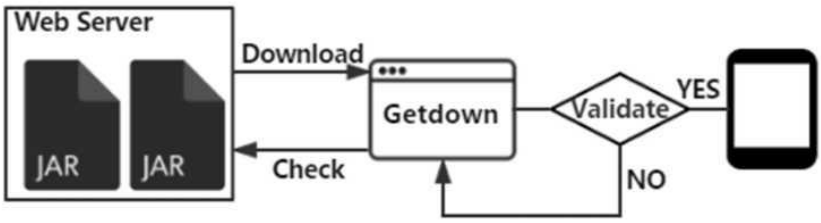

Рис. 1.9. Як працює програма Getdown

Принцип роботи Getdown показано на рис. 1.9. Процес Getdown спочатку прочитає вміст контрольних файлів, щоб визначити деякі необхідні параметри. Потім перевірте доступні файли та завантажте їх. Після завершення завантаження перевірте цілісність кожного цільового файлу на основі результату порівняння з дайджест-файлом. Якщо будь-який файл пошкоджено, спробуйте видалити завантажені файли та повторно завантажити всі файли, доки результат перевірки не буде хорошим. Згодом, якщо версія цільового файлу Jar вища за поточну версію, указану в контрольному файлі, перейдіть до процесу оновлення. В іншому випадку продовжуйте процес перевірки. Якщо процес проходить гладко, після встановлення оновлення перезапустіть програму, щоб запустити нову версію програми.

1.5 Типи мереж Інтернету речей

Мережі поділяються на категорії залежно від діапазону відстаней, які вони забезпечують. На рис. 1.10 наведено типи мереж IoT, що мають такі властивості та сфери застосування:

- Nano-мережа – набір невеликих пристроїв (розміром не більше кількох мікрометрів), які виконують дуже прості завдання, такі як виявлення, обчислення,

зберігання та приведення в дію. Такі системи застосовуються у біометричних, військових та інших нанотехнологіях. Nano-мережа може збирати важливу інформацію про пацієнтів у галузі охорони здоров’я та надавати її комп'ютерним системам, щоб зробити моніторинг здоров’я більш точним та ефективним. Окрім процесу виявлення пухлини, Internet Nano-Things в системах охорони здоров’я надаватиме діагностику та підтримку в лікуванні пацієнтів точними та локалізованими препаратами;

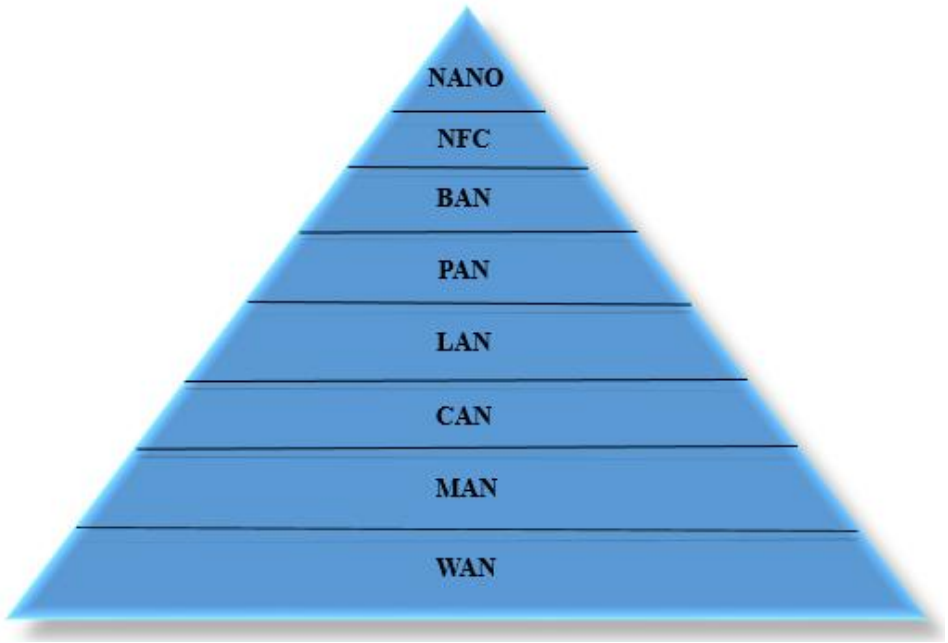

Рис. 1.10. Типи мереж IoT

- Near-Field Communication (NFC) – низько швидкісна мережа для підключення електронних пристроїв на відстані до 4 см один від одного.

Можливими областями застосування є безконтактні платіжні системи, документи, що засвідчують особу, та карти-ключі;

- Body Area Network (BAN) – мережа для підключення обчислювальних пристроїв, що носяться, які можна носити як закріпленими на тілі, так і поряд з тілом у різних положеннях, або вбудовувати всередину тіла (імплантати). BAN використовується в галузі медицини, покращення здоров'я, особистої безпеки та благополуччя, спорту та відпочинку;

- Personal Area Network (PAN) – мережа для з'єднання пристроїв у радіусі приблизно однієї або кількох кімнат. Прикладом є користувач, який подорожує з ноутбуком, приватним цифровим помічником та мобільним принтером, який може з'єднувати їх з бездротовою технологією, не потребуючи нічого підключати. Цей тип приватної мережі може також бути з'єднаний з Інтернетом або іншими мережами без кабелів;

- Local Area Network (LAN) – мережа, що покриває площу однієї будівлі. Комп'ютери та інші мобільні пристрої діляться ресурсами, такими як принтер або мережеве зберігання через локальне з'єднання. Wireless Fidelity (Wi-Fi) та Ethernet – це два основних способи включення LAN-з'єднань. Ethernet – це специфікація, яка дозволяє машинам взаємодіяти. Радіохвилі використовуються для підключення Wi-Fi комп'ютерів, принтерів, мобільних телефонів тощо.

Користувач має доступ до файлів, що зберігаються на локальному сервері разом з іншими; мережевий адміністратор має доступ для читання та запису;

- Corporate Area Network (CAN) – мережа, що об'єднує дрібніші локальні мережі в межах обмеженої географічної області (підприємство, університет);

- Metropolitan Area Network (MAN) – велика мережа для певного мегаполіса, що використовує технологію мікрохвильової передачі. Це дозволяє створювати розумні міста у всьому світі. Інтернет речей очолив розробку систем «розумного міста» для сталого способу життя, підвищення комфорту та продуктивності для громадян;

- Wide Area Network (WAN) – мережа, що існує на великомасштабній географічній території й об'єднує різні дрібніші мережі, включаючи LAN і MAN.

Прикладом є датчики, які використовуються для виявлення змін у фізичному та/або логічному зв'язку одного об'єкта з іншим та/або навколишнім середовищем. Фізичні зміни можуть включати температуру, світло, тиск, звук та рух. Логічні зміни включають наявність/відсутність об'єкта, розташування та/або діяльність, що відстежуються в електронному вигляді. У контексті IoT фізичні та логічні зміни однаково важливі. Ці типи рішень використовуються для багатьох промислових додатків у різних галузевих вертикалях.

1.6 Архітектура та протоколи Інтернету речей

Протоколи Інтернету є невід'ємною частиною стеку технологій Інтернету речей. Без протоколів і стандартів IoT апаратне забезпечення вважалося б марним. Це пов’язано з тим, що протоколи IoT дозволяють обладнанню обмінюватися даними. І з цих переданих фрагментів даних кінцевий користувач може отримати корисну інформацію.

Протоколи та стандарти IoT часто не враховуються, коли люди думають про IoT. Найчастіше індустрія приділяє увагу спілкуванню. І хоча взаємодія між пристроями, датчиками IoT, шлюзами, серверами та додатками користувача є важливими компонентами Інтернету речей, зв'язок буде порушений без правильних протоколів Інтернету речей.

Протоколи та стандарти IoT можна розділити на дві окремі категорії: - протоколи даних IoT (рівні представлення/додатки);

- мережеві протоколи для IoT (канал передачі даних/фізичний рівень).

Протоколи даних IoT використовуються для підключення IoT пристроїв з низьким енергоспоживанням. Вони забезпечують зв'язок з обладнанням на стороні користувача без необхідності підключення до Інтернету.

Архітектура IoT – це система з безлічі елементів, таких як датчики, приводи, протоколи, хмарні сервіси та рівні, що складають мережеву систему IoT.

Єдиного консенсусу з архітектури IoT немає, оскільки різні дослідники пропонували різні архітектури. Переважають три-, чотири- та п’яти рівневі архітектури.

Розглянемо класичну архітектуру Інтернету речей, що включає чотири рівні.

1) Сенсорний рівень. IoT-пристрої збирають показання з датчиків та виконують фізичні дії. Ці датчики або приводи приймають дані (фізичні параметри/параметри навколишнього середовища), обробляють дані та передають дані мережею.

2) Мережевий рівень. Шлюзи, які отримують інформацію від пристроїв та передають їм команди виконання дій. Як правило, представлені апаратним маршрутизатором чи ПЗ, які використовують різні протоколи. Розширені шлюзи, які

в основному відкривають з'єднання між сенсорними мережами та Інтернетом, також виконують багато базових функцій шлюзу, такі як захист від шкідливих програм, фільтрація, а також іноді прийняття рішень на основі введених даних та служб управління даними тощо.

3) Рівень обробки даних. Сервер, де зберігаються, обробляються та аналізуються показання датчиків перед посиланням до центру обробки даних, звідки доступ до даних здійснюється програмними програмами. Може бути реалізований на базі віртуального сервера, реальної машини або через хмарну технологію.

На рис. 1.11 зображено класичну еталонну модель IoT.

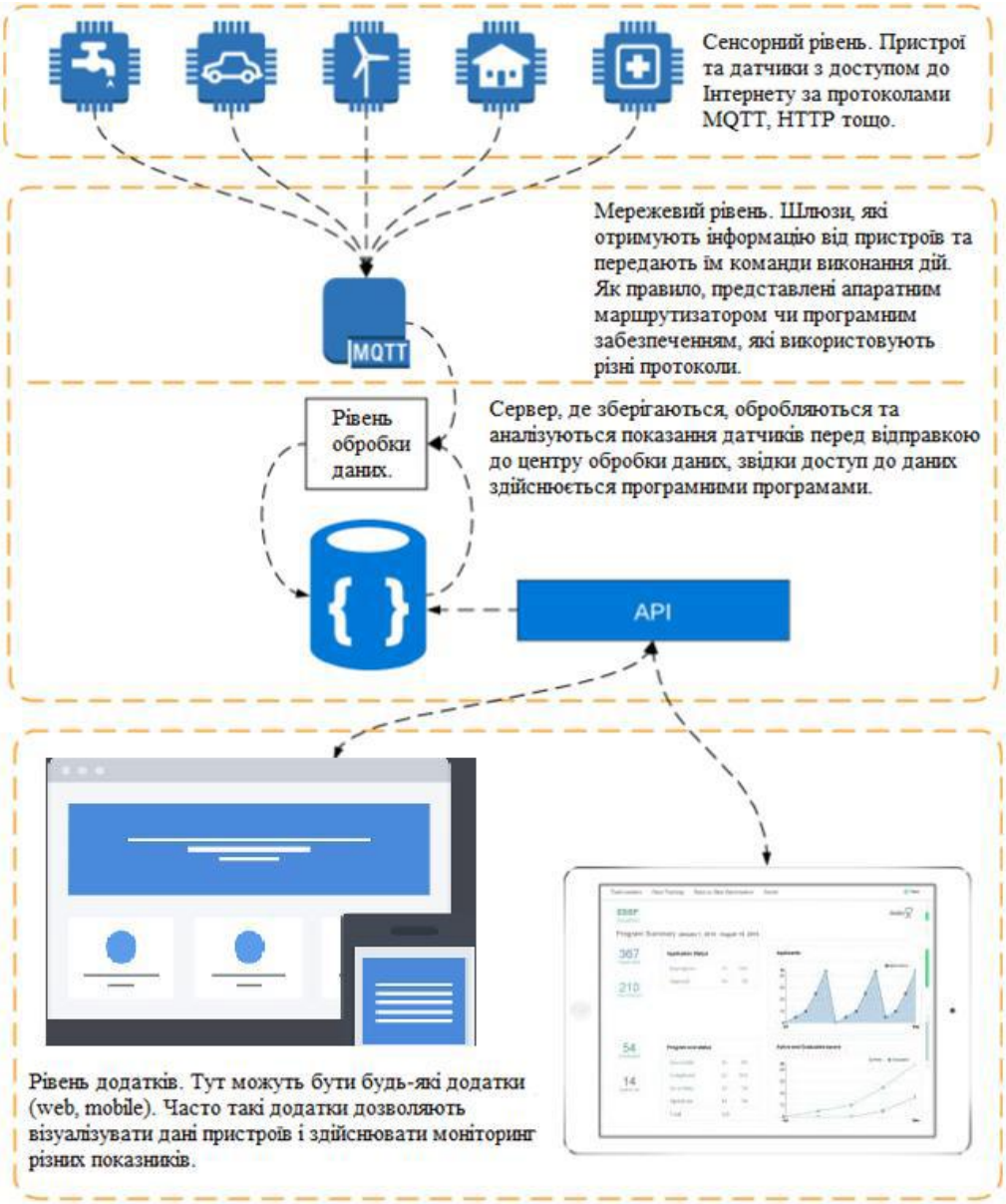

Рис. 1.11. Класична еталонна модель IoT

4) Рівень додатків. Клієнтська частина реалізується через мобільний або веб-додаток для управління даними. Забезпечує доступ до даних пристроїв та наочне представлення результатів аналізу.

Висновки до розділу

В даному розділі виконано аналіз предметної області побудови автоматичних оновлень в системах Інтернет речей. Наведені основні відомості про концепцію Інтернет речей, правила використання цифрового підпису в IoT. Описані процеси для автоматичного оновлення програмного забезпечення в системах Інтернету речей. Наведені архітектура та протоколи систем Інтернету речей.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ ТА МЕТОДОЛОГІЇ ПОБУДОВИ СИСТЕМ ІНТЕРНЕТУ РЕЧЕЙ

2.1 Розробка архітектури мережі

На основі вищевказаної мети дослідження треба визначити найбільш прийнятну комбінацію цих рішень з точки зору ступеня реалізації кожного рішення, складності технології впровадження, універсальності тощо. Після виконання цієї роботи розробіть базову теоретичну модель відповідно до комбінації рішень, яка ілюструє робочий процес між адміністратором, сервером, диспетчером пристроїв і пристроями.

Треба запропонувати сценарій підтвердження концепції та запровадити безпечну систему автоматичного оновлення на основі цього сценарію. Варто зазначити, що цей сценарій особливо повинен відображати високу масштабність і високу швидкість датчика. Виберіть відповідні методи впровадження та техніки для розробки цієї системи. Наприклад, у цьому проекті Maven Shade Plugin буде використано для створення одного файлу JAR. Адміністратор відповідає за підписання програми та завантаження нової версії програми до конкретної бази даних, базою даних буде сервер веб-сайту (сервер Tomcat). Більш того, програма Java буде використовуватися для моделювання концептуального сценарію, такого як адміністратор, пристрій і менеджер. Eclipse може надати інтегроване середовище розробки (IDE) майже для всіх мов і архітектур, тому виберіть Eclipse як інструмент розробки.

Перш за все, треба визначити систему відповідно до розрахованої загальної затримки та проміжної затримки та проаналізуйте ці результати, щоб перевірити, чи система має хорошу продуктивність. У той же час оцініть безпеку цієї системи відповідно до того, чи результати вимірювань відповідають очікуваним результатам, таким як відсоток успішного встановлення в різних умовах. А потім аналізувати ці

результати, даючи оцінку рівню безпеки. Нарешті підсумовано переваги та недоліки системи.

У цьому розділі спочатку розглядається загальна структура цієї системи автоматичного оновлення на рис. 2.1. Потім шар за шаром ілюструє більш конкретний робочий процес на рис. 2.2. Після вищезазначеної роботи описується детальний метод впровадження для кожного рівня цієї системи автоматичного оновлення відповідно до структури робочого циклу на рис. 2.2. Запропонована загальна структура мережі для схеми автоматичного оновлення показана на рис. 2.1.

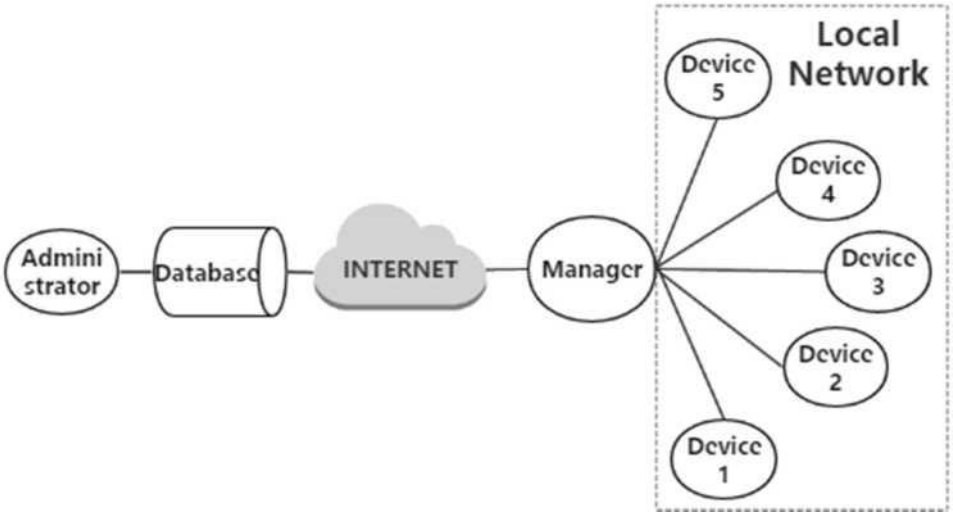

Рис. 2.1. Пропонована загальна архітектура мережі

Передбачається, що такий сценарій стався в промисловому Інтернеті речей. Мережа містить адміністратора, базу даних, Інтернет, менеджер і багато пристроїв. Менеджером зазвичай виступає шлюз локальної мережі, а пристроями часто є машина або пристрої керування, такі як Raspberry Pi. Коли заводу потрібно оновити якусь помилку, він просто завантажує нову програму в базу даних, а потім усі пристрої оновлюються автоматично.

Цей рис. 2.2 ілюструє зв’язок між адміністратором, базою даних, менеджером і всіма пристроями. По-перше, адміністратор буде безпосередньо взаємодіяти з базою даних. Ймовірно, менеджер буде взаємодіяти з базою даних безпосередньо. Тоді менеджер (зазвичай це шлюз, який може забезпечити доступ до Інтернету для

пристроїв локальної мережі [20]) буде спілкуватися з пристроями, якщо в базі даних є нова версія програми через Інтернет. Нарешті пристрій отримає нову версію програми з диспетчера пристроїв через локальну мережу. Вище наведено основний зв’язок між двома сутностями.

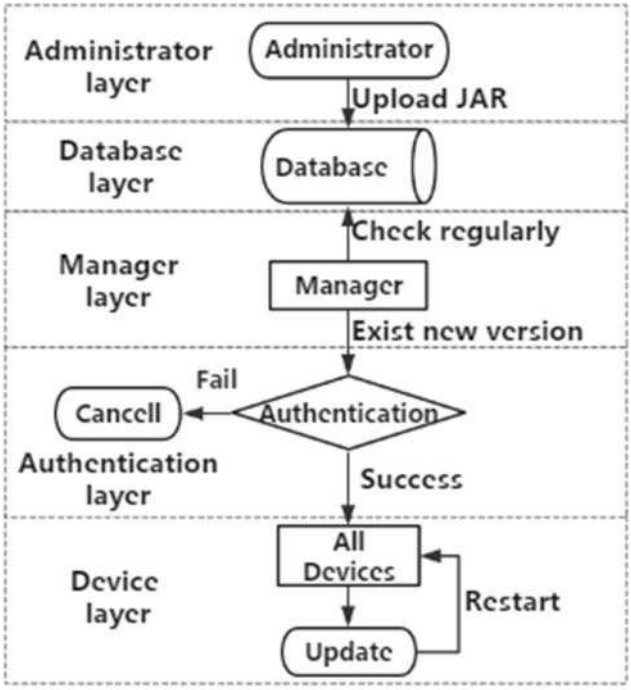

Рис. 2.2. Пропонований робочий процес комунікації в системі IoT

Більш детальний робочий процес комунікації показано на рис. 2.2. Спочатку адміністратор завантажить нову версію програми в базу даних. У той же час менеджер буде регулярно перевіряти наявність нової версії програми в існуючій базі даних. Якщо є нова версія програми, менеджер буде аутентифікувати пристрої один одного. Є два результати аутентифікації. Одним із результатів є помилка автентифікації, і в цьому випадку поточний процес оновлення певного пристрою буде припинено. Інший варіант — успішна автентифікація, після чого менеджер надішле нову версію файлу програми на всі локальні автентифіковані пристрої. Потім ці пристрої автоматично встановлять нову версію програми. Після завершення процесу інсталяції пристрій буде перезапущено та, нарешті, запущено нову версію програми.

2.2 Реалізація частини адміністратора. Опис основних функції сервера бази даних

Рівень адміністратора відповідає за розробку та реалізацію програми (JAR-файл), а також завантаження JAR-файлу до бази даних. Отже, у цьому розділі буде описано, як будуються додаток і адміністратор.

Перше, що потрібно реалізувати, це отримати правильну форму програми, яка буде передаватись у зв'язку та автоматично оновлюватися, і це має бути єдиний JAR. Щоб краще спостерігати за інформацією про інсталяцію програми, такою як хід виконання, номер версії, результати інстальованої програми показані нижче на рис. 2.3.

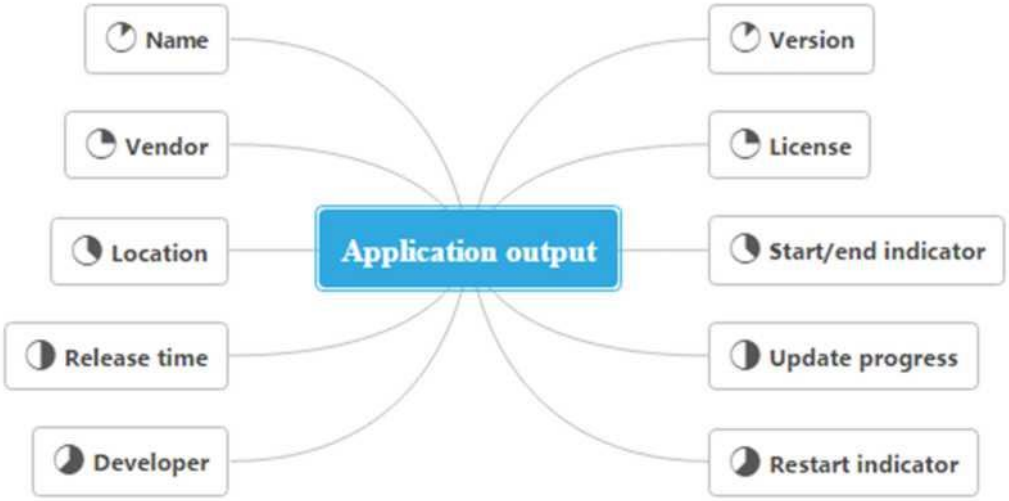

Рис. 2.3. Дизайн вихідних даних програми

Для зручності та ефективності цей проект вибере плагін Maven Shade [22] для створення єдиного файлу JAR. Плагін Maven Shade надає можливість запакувати проект у «виконуваний файл JAR». Специфікація виводу програми встановлюється згідно з рисунком 2.3.

Адміністратор — це інтерфейс користувача (інтерфейс користувача) програми для завантаження. Цей проект використовує SWT (Standard Widget Toolkit) для розробки інтерфейсу користувача адміністратора. SWT — це інструмент із

відкритим вихідним кодом, розроблений, щоб надати розробнику ефективний спосіб компонування віджетів. Наприклад, він може надати менеджери кнопок, міток, тексту та макета. Інтерфейс адміністратора складається з двох частин: інтерфейс дисплея та інтерфейс роботи.

Інтерфейс відображення: це відносно простий і містить лише три елементи відображення: назва програми, час випуску, поточна версія.

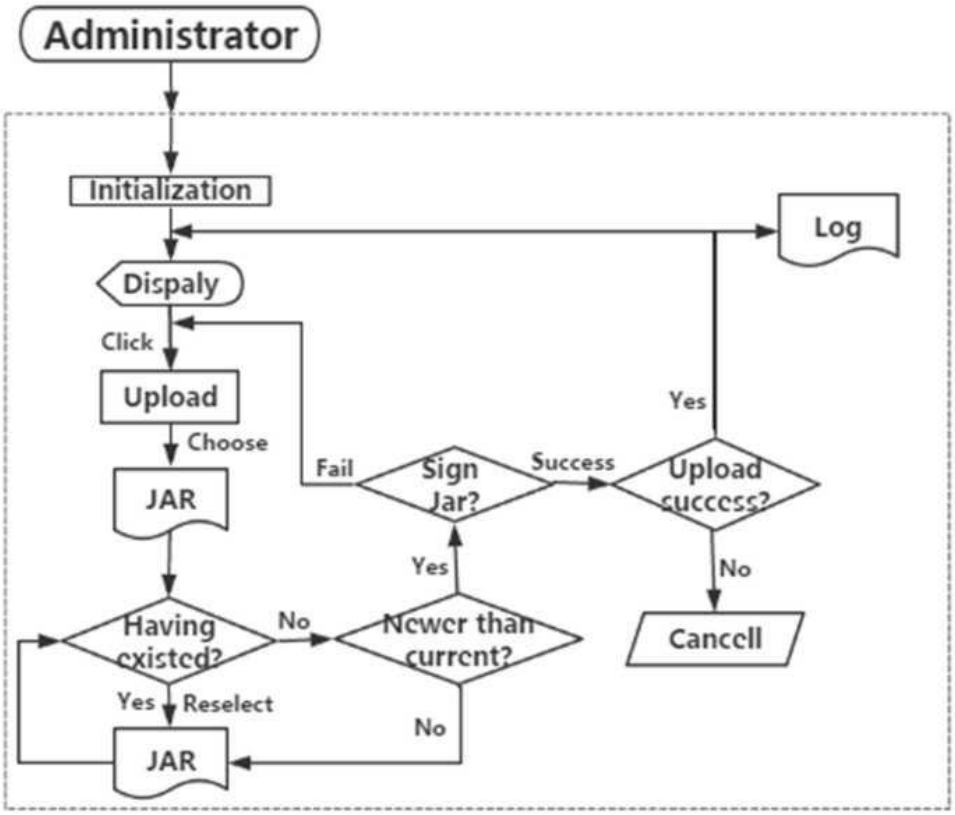

Рис. 2.4. Функціональна блок-схема адміністратора

Інтерфейс керування: цей процес показано на рис. 2.4. Операцію завантаження буде ініційовано натисканням кнопки «Завантажити». Потім програма запропонує вікно вибору для вибору файлу JAR. Якщо вибраний файл є існував на сервері бази даних, а потім просто запропонуйте повторно вибрати. Якщо ні, він продовжить оцінювати, чи вибрана версія програми є новішою за поточну версію (номер версії вибраної програми більший за номер поточної версії). Якщо ні, подібно до останнього суду існуючого, він також запропонує повторно вибрати. Якщо він новіший, то програма підпише файл JAR за допомогою інструменту Jarsigner, щоб

забезпечити цілісність файлу JAR. Якщо підписати успішно, виконайте фактичну команду "завантажити". Якщо завантаження не вдається, операцію завантаження буде скасовано. У разі успіху підписаний файл JAR буде надіслано на сервер і збережено в базі даних. Водночас адміністратор реєструватиме інформацію про завантажену програму.

Цей рівень бази даних реалізовано через Apache Tomcat, також званий Tomcat Server. Сервер Tomcat розроблений Apache Software Foundation (ASF) [23]. Основна мета сервера Tomcat — допомогти користувачеві швидко розробити веб-програму. Веб-програма не схожа на звичайну окрему програму, оскільки вона працює в Інтернеті, наприклад Amazon, Google.

На рисунку 2.5 показана структура базової функції завантаження та завантаження файлу на сервері Tomcat. Ці функції є частиною впровадження системи автоматичного оновлення. У цій конструкції є дві сторони: сторона клієнта та сторона сервера. Клієнтська сторона містить адміністратора та менеджера.

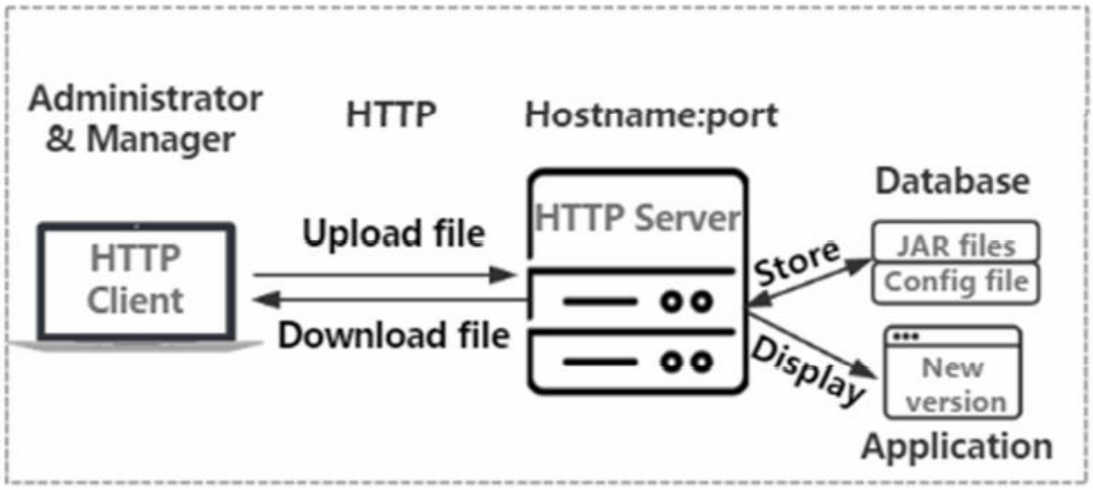

Рис. 2.5. Основні функції сервера бази даних

Сторона сервера відображається у формі «ім’я хоста: порт» і містить базу даних і програму. База даних складається з двох частин: колекції JAR-файлів і одного конфігураційного файлу, який вказує на поточну найновішу версію. Різниця та інновація між цим Проект і звичайна система автоматичного оновлення полягає в тому, що клієнту не потрібно шукати всі файли програми один за одним замість

того, щоб перевіряти вміст версії конфігураційного файлу.Програма використовується для відображення інформації про нові файли у візуальний спосіб.

По-перше, сторона клієнта (адміністратор) ініціює запит, який містить завантаження файлу Jar на сервер. Після того як сервер отримає його, він збереже файл Jar у своїй базі даних і відповість клієнту, щоб стверджувати, що сервер отримав і зберіг його. У той же час програма сервера відображатиме найновішу інформацію про програму. По-друге, менеджер ініціює URL-запит до файлу конфігурації бази даних, щоб перевірити, яка найновіша версія в базі даних, і вирішити, чи потрібно завантажити новий файл Jar. Якщо так, то менеджер завантажить його з сервера.

Рівень менеджера є ядром цієї системи автоматичного оновлення. Він з'єднує базу даних і всі пристрої як міст. З точки зору функції, Менеджер має дві основні функції: завантажити JAR-файл (Downloader) і відправити його на пристрої (Sender). У цьому розділі буде описано, як менеджер реалізує ці дві частини згідно з рисунком 2.6.

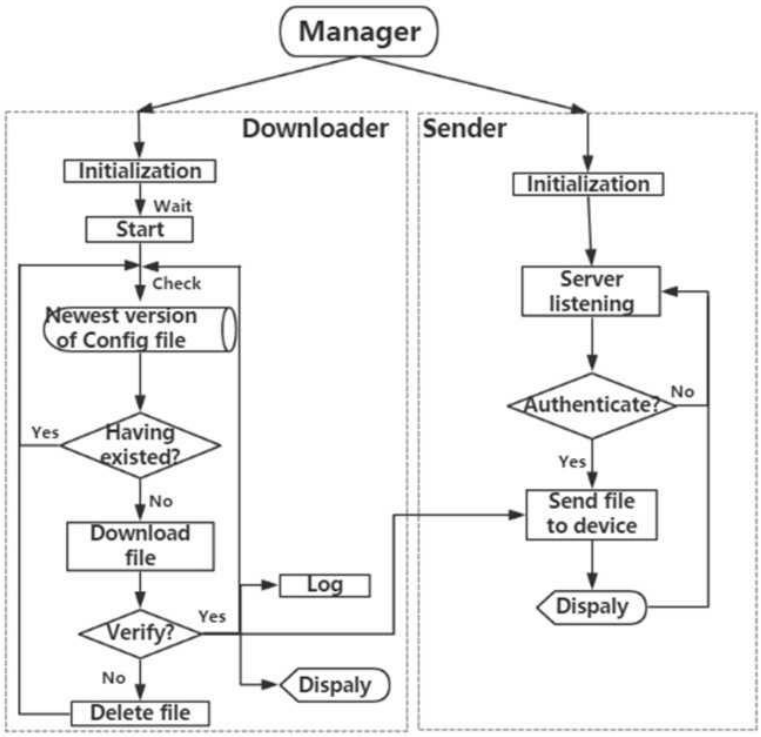

Рис. 2.6. Функціональна блок-схема менеджера

Завантажувач призначений для завантаження найновішого JAR-файлу бази даних. Функцію завантажувача показано в лівій частині рисунка 2.6. Робочий процес завантажувача можна розділити на три етапи: етап підготовки, етап завантаження, етап перевірки.

Етап 1, завантажувач ініціює необхідні параметри для наступної серії завдань на етапі підготовки, такі як робочий каталог, URL веб-сервера тощо. Тоді, коли розпочнеться процес перевірки, він перевірятиме номер версії файлу конфігурації в базі даних кожного періоду часу. Якщо у сховищі завантажувача існувала найновіша версія файлу JAR, він просто ігнорує його та чекає перевірки наступного ходу. Якщо ні, то він перейде до етапу завантаження.

Етап 2, етап завантаження, завантажує найновіший файл JAR, указаний у файлі конфігурації, за допомогою протоколу HTTP. І збережіть цей файл JAR у своєму репозиторії.

Етап 3: після завантаження файлу JAR програма завантаження перевірить його цілісність на випадок, якщо файл пошкоджено, але його ще потрібно запустити. Більш детальне пояснення процесу перевірки буде обговорюється в розділі 2.2. Якщо він пошкоджений через певні причини, тоді він видалить цей JAR-файл і всі пов’язані файли та повернеться до процесу перевірки етапу 1. Якщо він не пошкоджений, то завантажувач запише файл журналу, щоб записати інформацію про завантаження цього часу. , як-от час завантаження, назва та версія програми тощо. Крім того, він оновить дисплей, на якому завжди відображається інформація про найновішу версію. У той же час це ініціює відправника надіслати файл на пристрій.

Відправник використовується для реалізації фактичної функції надсилання файлів. Його функції показано на правій стороні рисунка 2.6. Робочий процес відправника також розділений на три етапи: етап підготовки, етап автентифікації та етап надсилання.

Етап 1, як і завантажувач, перше, що потрібно зробити на етапі підготовки, це ініціювати процес, наприклад створити sslsocket сервера та встановити деякі параметри. Потім сервер почне слухати і чекати, поки клієнт (пристрій)

підключиться до нього. Якщо будь-який пристрій запитує підключення до сервера, то він переходить до етапу 2.

Етап 2, коли сервер успішно підключається до клієнта, тоді менеджер автентифікується за допомогою цього пристрою з питань безпеки. Більш детальний процес автентифікації буде розглянуто в розділі Якщо автентифікація не вдається, він повернеться до процесу прослуховування етапу 1. В іншому випадку він перейде до етапу 3.

Етап 3, функція остаточного відправлення залежить від двох факторів: результату перевірки та аутентифікації. Лише коли обидва ці два фактори вірні, функція надсилання може бути виконана. Після надсилання файлу відправник оновить вміст інтерфейсу користувача менеджера та відобразить інформацію про новішу версію програми.

2.3 Опис процесу автентифікації. Використання цифрового підпису

Рівень автентифікації дуже важливий для цієї системи, оскільки він не має сенсу для системи без гарантії безпеки. На рис. 2.7 представлено загальні методи безпеки цієї системи автоматичного оновлення. Можна побачити, що методи безпеки поділяються на три категорії: SHA, DSA та SSL.

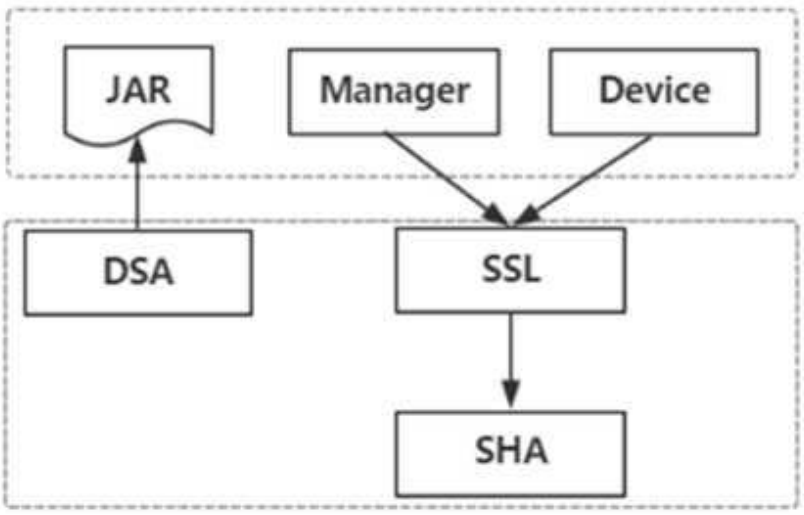

Рис. 2.7. Загальні методи безпеки пропонованої системи

Основне призначення цифрового підпису використовується для перевірки цілісності JAR-файлу в цій системі. Після створення JAR-файлу його потрібно підписати. Це може запобігти запуску пристрою пошкодженого JAR і призвести до непередбачуваних результатів. На рис. 2.8 пояснюється, як ця система автоматичного оновлення реалізує цифровий підпис і перевірку.

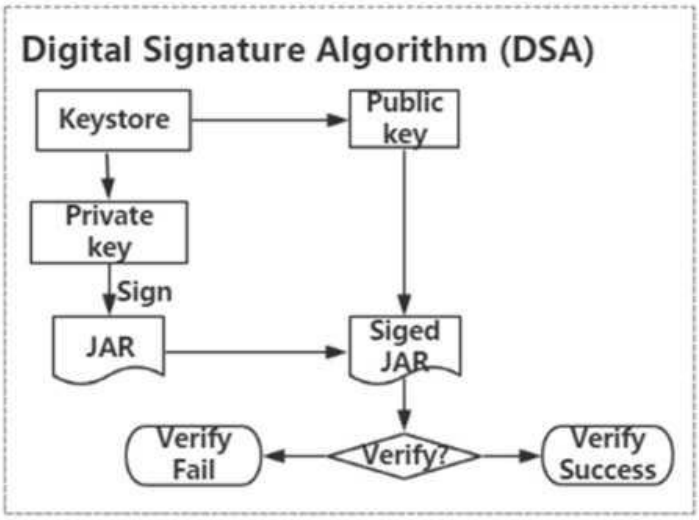

Рис. 2.8. Потік роботи DSA для системи автоматичного оновлення

Спочатку створіть сховище ключів за допомогою інструмента Keytool, щоб створити закритий та відкритий ключ (експортований у сертифікат). Потім підпишіть файл JAR за допомогою інструмента Jarsigner із закритим ключем. Jarsigner — це інструмент підпису на основі Java KeyStore (JKS), який є сховищем сертифікатів безпеки. Інструмент Jarsigner має дві функції: підписати файл JAR і перевірити підписаний файл JAR. Відповідний сертифікат використовується для перевірки підписаного файлу JAR.

Підпис: це відбувається, коли адміністратор вирішує завантажити файл JAR, програма підпише JAR за допомогою закритого ключа, указаного розробником.

Перевірка: сертифікат надсилається до менеджера та пристрою, які використовуватимуться для перевірки підпису. Існує два випадки перевірки файлу JAR. По-перше, після того, як менеджер завантажить JAR, він виконає першу перевірку. Причиною цього є те, що JAR-файл може бути підроблено туристом у

базі даних або JAR-файл було пошкоджено під час завантаження. По-друге, після того, як пристрій отримає файл JAR, він виконає повторну перевірку. Для цього файл JAR може бути пошкоджений під час надсилання та отримання.

Звичайний сокет забезпечує лише мережевий зв’язок без гарантії безпеки. Але сокет SSL може компенсувати обмеження для реалізації безпечного мережевого зв’язку. Основний принцип SSL-сокета для досягнення безпеки проілюстровано на рис. 2.9 [24].

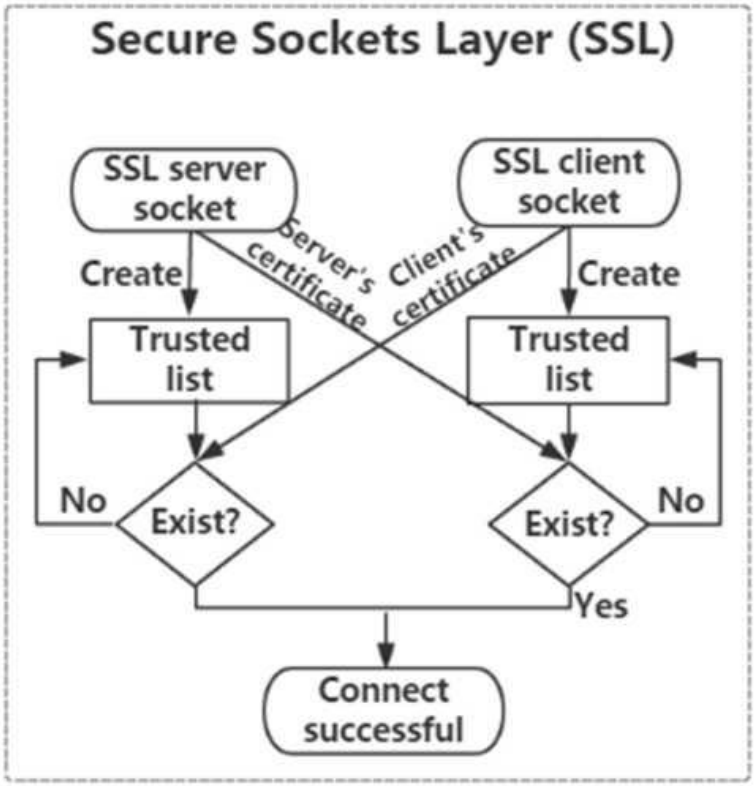

Рис. 2.9. Робочий процес SSL для системи автоматичного оновлення

Перед встановленням з’єднання сервер (Менеджер) і клієнт (Пристрій) повинні мати два файли, пов’язані із сертифікатом. Один файл є власним сертифікатом, інший файл є довіреним списком, який записує сертифікати, яким він довіряє. Коли серверний сокет SSL починає слухати, а клієнтський сокет SSL запитує підключення, обидва надсилають власний сертифікат іншій стороні. Коли вони отримають сертифікат, вони здійснять пошук у всьому списку надійних, щоб перевірити, чи існує сертифікат іншої сторони. Якщо ні, просто відмовтеся від цього

з’єднання та перейдіть до наступного ходу. Якщо обидва сертифікати існував у довіреному списку іншої сторони, тоді сервер і клієнт встановлять реальне з’єднання.

2.4 Представлення алгоритму безпечного хешування

Автентифікація SHA запускається лише тоді, коли сокет SSL підключається успішно, як показано на рис. 2.7. Це подвійна безпека, яка може значно підвищити рівень безпеки цієї системи, а рис. 2.10 описує це детально.

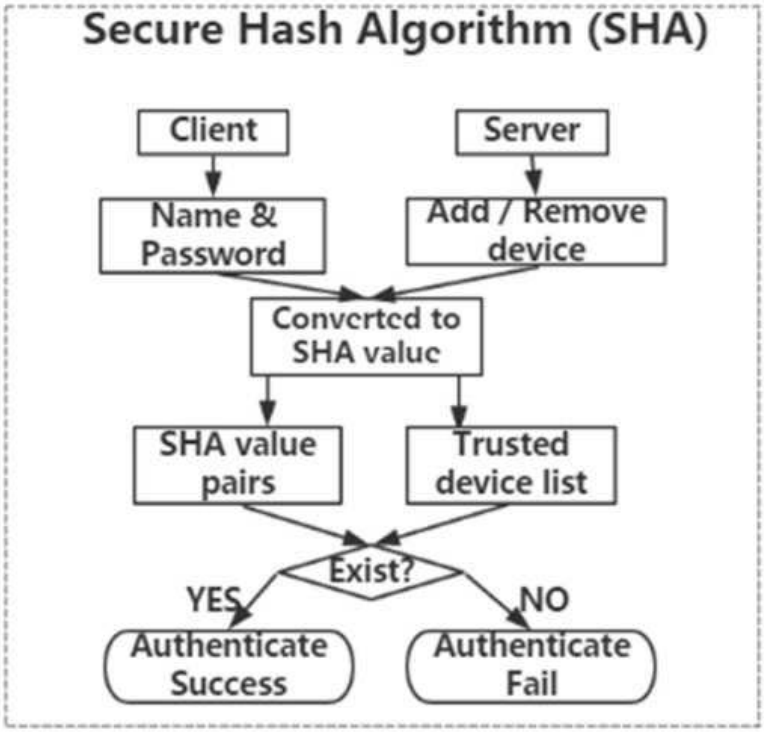

Рис. 2.10. Робочий процес SHA для системи автоматичного оновлення

Сервер (Менеджер) має список довірених пристроїв (відображається у вигляді файлу) і підтримує його, додаючи та видаляючи пристрої. Припустимо, що кожен пристрій має дві властивості: ім’я та пароль. Коли сервер додає пристрій, він зберігатиме ім’я та значення SHA1 пароля до списку надійних пристроїв. Під час видалення пристрою просто видаліть ім’я та значення SHA1 пароля зі списку надійних пристроїв. Причиною цього є те, що файл і повідомлення від клієнта

можуть бути скомпрометовані мандрівником, тому ім’я пристрою та його відповідний пароль можуть бути розкриті. У результаті турист може вчинити зловмисний спосіб, позуючи пристрій. Але якщо зберегти та надіслати значення SHA пароля, цих поганих результатів можна уникнути. Після того як сервер і клієнт встановлять з’єднання через сокет SSL, клієнт надішле серверу своє ім’я та значення SHA пароля. І сервер порівнює його з парою ключ-значення у своєму списку надійних пристроїв після отримання повідомлення від клієнта. Якщо існує пара ключ-значення, кінцевим результатом автентифікації є успіх. Інакше автентифікація не вдасться.

Рівень пристрою є останнім рівнем цієї системи автоматичного оновлення. З функціональної точки зору пристрій має дві основні функції: отримання файлів JAR (приймач) і їх встановлення (інсталятор). Наступні два розділи йдуть за малюнком 2.11, щоб описати, як пристрій реалізує ці дві функції.

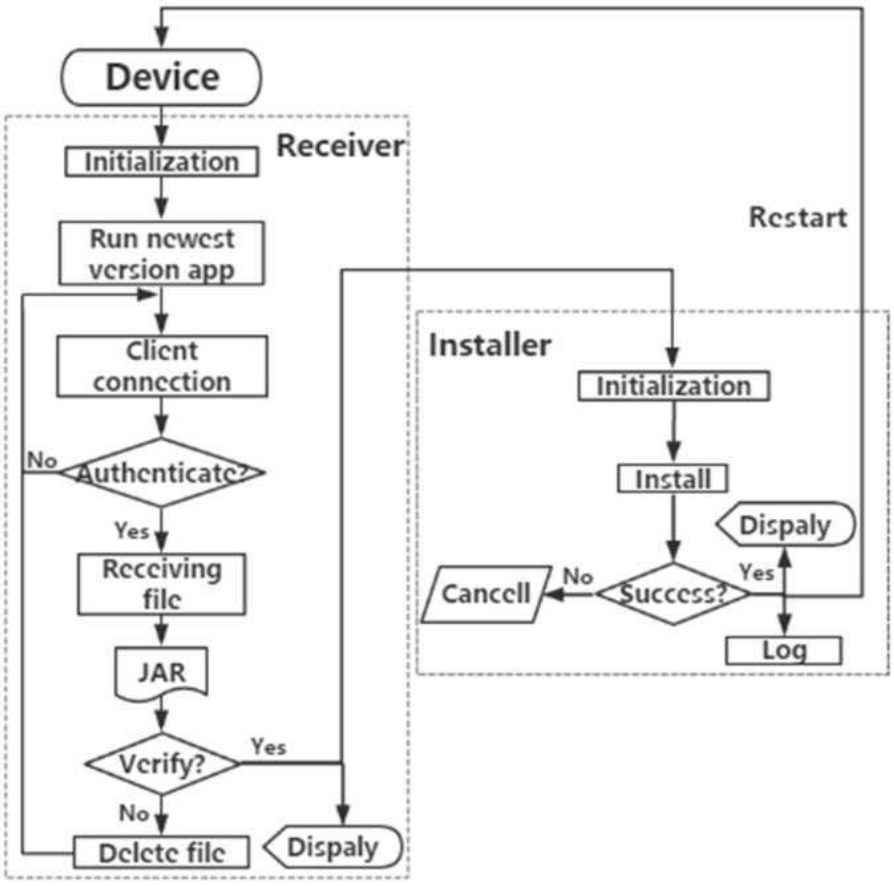

Рис. 2.11. Функціональна блок-схема пристрою

Призначення приймача — отримати JAR-файл від завантажувача Manager. Ця частина показана в лівій частині рисунка 2.11. Відповідно до основної функції приймача, процес можна підсумувати як три етапи. Стадія підготовки, стадія отримання та стадія перевірки.

Етап 1, етап підготовки, охоплює кілька підпроцесів. По-перше, як і звичайний процес, він ініціює необхідні параметри, такі як робочий каталог. І запустіть поточну програму, яка є найновішою версією. Варто зазначити, що програма завжди працюватиме після запуску до встановлення нової програми. Потім приймач шукає конкретний сервер і запитує підключення. Коли підключення успішне, перейдіть до оцінки умови «Автентифікація», якщо його пройдено, перейдіть до етапу отримання, інакше просто поверніться назад.

Етап 2, на етапі отримання приймач отримає файл JAR від завантажувача через мережу TCP/IP і збереже його в локальному місці пристрою. Це критично важливий крок, оскільки метою попередньої роботи є успішне отримання файлу JAR без втрат.

Етап 3, перевірка відбувається після отримання JAR-файлу, метою якої є перевірка цілісності JAR-файлу. Якщо файл JAR пошкоджено, видаліть усі отримані файли, відновіть середовище та поверніться до етапу 1. Якщо ні, потрібно зробити дві речі. Відображення деякої інформації про отримання файлу та запуску інсталятора.

Кінцевою метою цієї системи автоматичного оновлення є успішне встановлення нової версії програми. Хід процесу показано в правій частині рисунка 2.11. Інсталятор запускається після отримання вказівки від приймача. Інсталятор також містить три етапи: етап підготовки, етап встановлення та етап перезапуску.

Етап 1, умовно кажучи, підготувати етап завдання інсталятора легко. Програмі встановлення потрібно лише запустити процес, наприклад джерело файлу JAR, розташування файлу журналу тощо.

Етап 2, етап інсталяції, надзвичайно важливий, оскільки на результати інсталяції впливає багато факторів, таких як стан робочого середовища, обсяг пам’яті тощо. Цей крок, який перевіряє, чи запущена відповідна програма перед фактичним встановленням, необхідний. Якщо існує, інсталятор знищить його

відповідними способами. Після того, як вищезазначене завершено, інсталятор виконує командний рядок «java -jar xxx.jar» у фоновому режимі, щоб інсталювати файл JAR. У разі успіху перейдіть до етапу 3. В іншому випадку вийдіть із процесу встановлення.

Етап 3, після встановлення нової версії програми необхідно скинути запущене середовище, тому необхідний перезапуск пристрою. Після виконання команди «Перезавантажити» і пристрою знову увімкнеться, пристрій запустить програму нової версії.

2.5 Метод застосування REST API в мережі IoT

REST API використовуються як проміжне програмне забезпечення, тобто виступає в ролі сервера між клієнтом та сховищем даних. REST є збіркою правил для стандартизованого зв’язку між клієнтом та сервером.

Щоб система вважалась сконтрюйованою по REST архітектурі необхідно, шоб вона задовільняла наступні критерії.

1) Client-Server. Система повинна бути розподілена на клієнтів та серверів. Розподілення інтерфейсів означає, що наприклад клієнти не зв’язані напряму з сховищем даних, яке залишається всередені кожного сервера, так щоб покращилась мобільність кода клієнта Сервери не пов’язані з інтерфейсом користувача, отже сервери можуть бути простішими та масштабуючими. Сервери і клієнти можуть бути взаємозамінюючими та розроблятися окремо.

2) Stateless. Сервер на має зберігати інформацію про клієнтів . В запиті повинна передаватися вся необхідна інформація для його обробки і ідентифікація клієнта.

3) Cache. Кожна відповідь має бути відмічена чи є чи потрібно його поміщати в кеш, або ні, для того щоб ми могли попередити повторне використання клієнтами застарілих або некоректних даних в відповідь на подальші запити.

4) Uniform Interface. Єдиний інтерфейс визначає інтерфейс між клієнтом та сервером. Це спрощує та відділяє архітектуру, яка дозволяє кожній частині розвиватися самостійно. Чотири принципа єдиного інтерфейсу:

– Identification of resources (заснований на ресурсах). В REST ресурсом являється усе те чому ми можемо надати ім'я, або ідентифікатор. Наприклад, користувач, зображення, предмет. Кожен ресурс в REST повинен бути ідентифікований за допомогою стабільного ідентифікатору, який не змінюється при зміні стану ресурса. Ідентифікатором в REST являється URI.

– Manipulation of resources through representations. (Маніпуляція над ресурсами через представлення). Представлення в REST використовується для виконання дій над ресурсами. Представлення ресурса представляє собою поточний або бажаний стан ресурса. Наприклад, якщо ресурсом являється користувач, то представленням може являтися XML або HTML опис цього користувача.

– Self-descriptive messages (само-опис повідомлення). Під самоописом мається наувазі, що запит і відповідь повинні містити в собі всю необхідну інформацію для їх обробки. Не повинні бути додаткові повідомлення або кеш для обробки одного запиту. Іншими словами відсутність стан, сохраняємого між запитами до ресурсів. Це дуже важно для масштабування системи.

– HATEOAS (hypermedia as the engine of application state). Статус ресурсу передається через інформацію, що міститься в body, параметри запиту, заголовки запитів та потребуючий URI (ім'я ресурсу). Це називається гіпермедіа HATEOAS також означає, що, в випадку необхідності силки можуть міститися в тілі відповіді (або заголовках) для підтримки URI.

5) Layered System. В REST допускається разділити систему на ієрархію слоїв але з умовою, що кожен компонент може бачити компоненти тільки безпосередньо наступного слою. Наприклад, якщо ви викликаєте службу PayPal, а він в свою чергу викликає службу Visa, про виклик служби Visa нічого не повинні знати.

6) Code-On-Demand (опціонально). В REST дозволяється загрузка та виконання коду якщо програми на стороні клієнта.

Сервери можуть тимчасово розширяти або кастомізувати функціонал клієнта, передаючи йому логіку, яку він може виконувати. Наприклад, це можуть бути скомпільовані Java-апплети або клієнтські скрипти на Javascript

REST API дозволяють відкрити підключений пристрій користувачам у програмі безпечним способом. RESTful API широко використовуються в сучасній мережі. Передача даних зазвичай здійснюється за допомогою JSON або XML через HTTP. Це гарна модель для неоднорідних систем. REST API робить інформацію про пристрій легкодоступною. Вони можуть стандартизувати спосіб створення, читання, оновлення та видалення даних. Усі ці операції включені до REST-запиту [16]. API REST дозволяють делегувати та керувати авторизацією. API може автентифікуватись на сервері і сервер можуть аутентифікувати API, щоб запобігти атаки в середині [5].

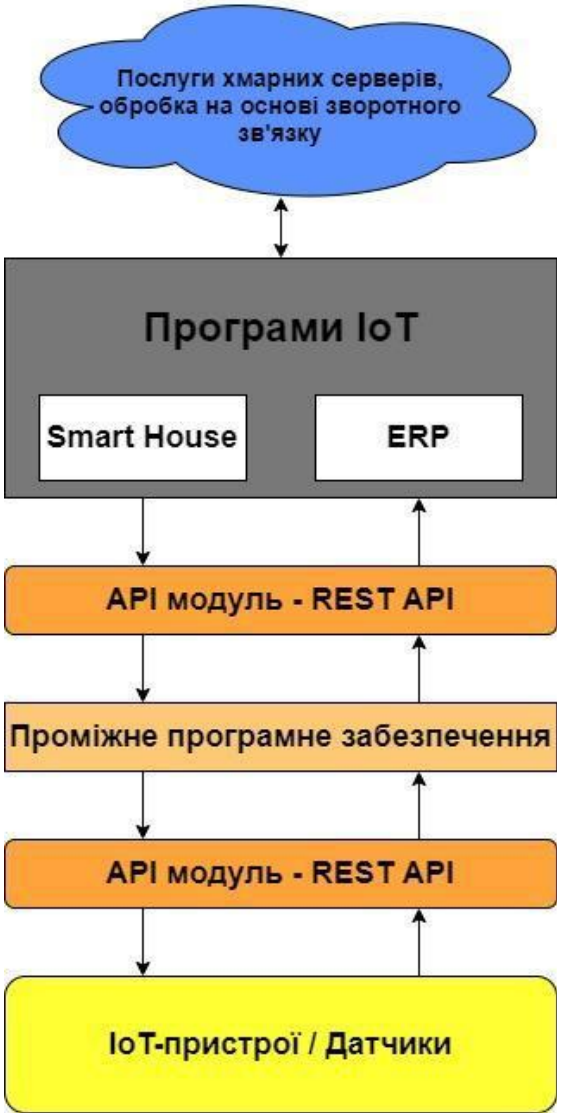

Рис. 2.12. Модель архітектури проміжного програмного забезпечення в мережі IoT

На рис. 2.12 представлена загальна картина ролі проміжного програмного забезпечення в IoT. У загальному сенсі маємо чотири категорії основних компонентів системи IoT - датчики, локальна мережа, яка може включати шлюз, проміжне програмне забезпечення, хмарне сховище [1].

Доступно багато протоколів авторизації. Наприклад, OAuth - це відкритий протокол авторизації, який може надати доступ до ресурсів через програмне забезпечення, використовуючи ім'я користувача, пароль та токени.

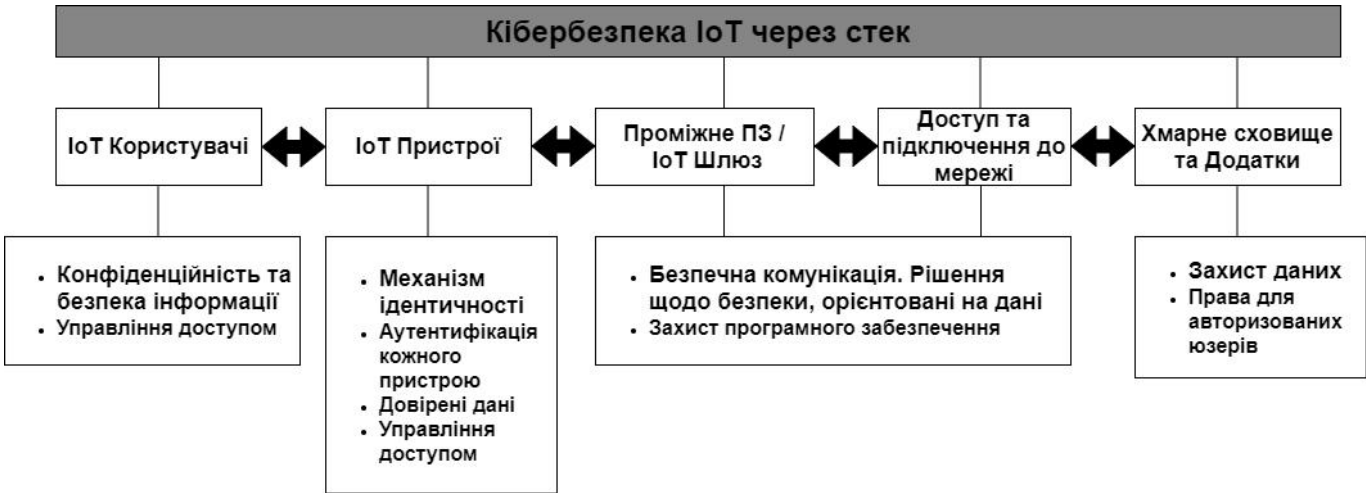

Рис. 2.13. Модель захисту для системи Інтернету речей

Протокол обмеженого застосування:

– Частина IPV6 для обмежених ресурсних середовищ;

– Інтерфейс API REST, орієнтований на IoT, відображається на ефективний бінарний протокол;

– Кінцеві пристрої можуть мати як клієнтську, так і серверну ролі, також можуть безпосередньо спілкуватися;

– Представлення ресурсів, наприклад 'application/json' , узгоджується як у протоколі http;

– Дотримуйтесь опції для тривалої реакції потоку подій на GET, асинхронні оновлення;

– Веб-зв'язок стандартного формату основного посилання IETF;

– Виявлення ресурсів за допомогою /well-known/core або зовнішнього каталогу ресурсів HTTP/REST:

– HTTP та REST корисні для більшості веб-додатків, служб та шлюзів;

– Використовує веб-виклики, веб-сокети, HTTP PUT для асинхронних оновлень;

– Посилання в JSON, Link-format, Hypercat, пов'язані між собою дані;

– Веб-посилання для IoT.

– Інкапсуляція об'єктів: об'єкти, що описують себе, мають посилання, пов'язані з кінцевими точками, наприклад /well- known/core;

– Каталог ресурсів: дерикторії та каталоги посилань, що зберігаються у відомих місцях, що вказують на ресурси та інші каталоги;

– Посилання можуть бути зареєстровані в Каталозі ресурсів, коли сенсори під'єднані до мережі;

– Сканер може знаходити посилання за адресами /well-known/core та заповнювати каталоги;

– Прикладне програмне забезпечення може виявляти ресурси за атрибутом, наприклад запит, використовуючи type = 'temperature', units = 'celsius', location = 'outdoors';

– Система повертає набір посилань, що відповідають атрибутам запиту.

2.6 Модифікований метод захисту інформації в IoT

Для модифікації методу який базується на REST API для підвищення рівню безпеки інформації в мережі, я пропоную використання криптографічного алгоритму еліптичних кривих. Протокол Діффі-Хеллмана на еліптичних кривих — криптографічний протокол, який дозволяє обом сторонам, що мають сполучення відкритий/закритий ключ на еліптичній криптографії, отримати спільний секретний ключ, використовуючи незахищений від загроз шлях зв'язку. Цей секретний ключ може бути застосований як для шифрування подальшого обміну, так і для формування нового ключа, який потім може використовуватися для подальшого

обміну інформацією за допомогою методів симетричного шифрування. Це варіант протоколів з використанням еліптичних кривих [15]. Опис алгоритму: Нехай абоненти: Клієнт та Сервер. Наприклад, Клієнт створює спільний секретний ключ з Сервером, але доступний між ними канал може бути прослуховуваним зловмисниками. Спершу повинен узгодитися набір параметрів (поля характеристики, загальні випадки). Відповідно у кожної зі сторін повині бути ключі, що складаються з секретного ключа , та публічного ключа. Більша кількість протоколів, які основані на ECDH, використовують функціонал процесу формування ключів для отримання значення симетричного ключа . З цієї інформації, зв'язаної зі своїм секретним ключом, повідомляється Клієнт тільки про свій публічний ключ.

Отже окрім клієнту ніхто не зможе обрахувати її секретний ключ, окрім пристрою який має права вирішити задачу дискретного логарифмування на еліптичних кривих. Секретний ключ Серверу також захищений. Ніхто окрім клієнту або серверу не може обчислити їх спільний секретний ключ, окрім учасника який має права вирішити проблему Діффі — Геллмана. Публічні ключі: статичними або ефемерні. Ефемерні ключі застосовуються лише на конкретний проміжок часу та не автентифікують абонента, отже, якщо потребується авторизація, верифікація автентичності має біти отриманим іншим методом. Якщо Клієнт або Сервер мають спільний статичний ключ, вразливість атаки посередника не враховується, не може забезпечуватися ні прямий секрет, ні стійкість підміни під час декриментації ключів, також інші параметри стійкості до вразливостей. Користувачі секретних ключів повинні перевіряти інший публічний ключ і застосовувати функцію процесу формування ключів на спільний секретний ключ, для того щоб виключити витоків інформації про статично секретний ключ [14]. Для кодування з іншими параметрами часто застосовують протокол MQV(Menezes–Qu–Vanstone).

При застосуванні спільного секретного ключу, часто бажано кешувати секретну інформації, для того щоб позбутися від атак, що з'явилися після використання протоколу.

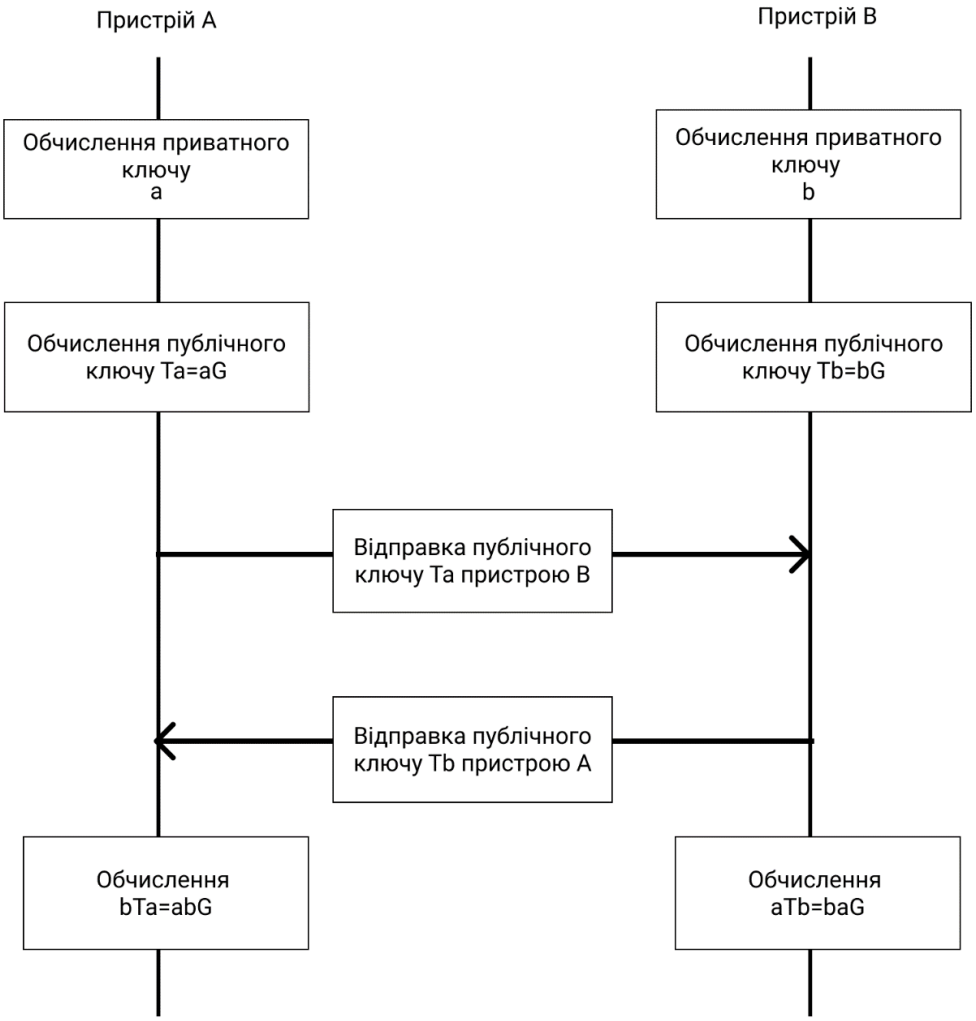

Рис. 2.14. Блок-схема роботи алгоритму еліптичної криптографії

Для реалізації на серверній частині цього алгоритму використано бібліотеку “crypto” для формування публічного та секретного ключу.

Для здобуття цілі підвищення захисту в інформації в мережі інтернету речей я пропоную такий алгоритм:

1) Розробка REST API та дерева URL для запитів з клієнту.

2) Розробка програмного забезпечення для клієнту.

3) Розробити логіку запитів.

4) Формуємо на сервері публічний ключ пристрою за допомогою Протоколу Діффі-Хе́ллмана на еліптичних кривих використовуючи будь який тип кривих та кодуємо його в типі base64.

5) При відправленні інформації на сервер спочатку мі повинні запит на сервер для отримання Публічного ключу який зберігається на сервері.

6) В тілі запиту ми маємо відправляти ідентифікатор пристрою.

7) Запит оброблюється на сервері, формується тіло відповіді серверу на запит, в тілі відповіді в нас передається публічний ключ в зашифрованому виді. Шифрується цей ключ за допомогою симетричного алгоритму блочного шифрування «AES256» в якому ключом до шифру виступає ідентифікатор. Таким чином ми добиваємося того, що при кожному запиті на сервер вертатися в нас буде зашифрований публічний ключ серверу і кожен раз він буде різний.

8) На стороні клієнту за допомогою ключу шифрування ми розшифровуємо публічний ключ серверу.

9) На стороні клієнту формуємо публічний ключ для даного запиту за допомогою Протоколу Діффі-Хе́ллмана на еліптичних кривих використовуючи будь який тип кривих та кодуємо його в типі base64.

10) Наступним кроком є формування секретного ключу пристрою для даного запиту. Формуватися секретний ключ буде на основі публічного ключу сервера, який ми отримали раніше.

11) Для коректної передачі даних ми повинні їх конвертувати до типу даних string.

12) Кодуємо дані які ми хочемо передати за допомогою алгоритму “AES256” в якому ключем шифрування виступає секретний ключ клієнта який ми сформували раніше.

13) Формуємо змінну яку ми будемо передавати в тілі запиту вона буде формуватися з рядку в якому буде в одному рядку публічний ключ, та в якості додаткових розрядів буде виступати публічний ключ зашифрований за допомогою AES256 в якості ключа до якого буде виступати публічний ключ серверу.

14) Робимо запит в тілі якого буде передаватися об’єкт з двома полями Поле 1 це змінна яку ми формували на кроці 13 Поле 2 зашифрованні дані які ми формували на кроці 12.

15) На сервері Розшифровуться Поле 1 та ми отримуємо публічний ключ клієнта.

16) На основі отриманого публічного ключу клієнту ми формуємо секретний ключ серверу, що формує між клієнтом та сервером сеансову пару ключів в ході чого секретний ключ сформований на клієнті буде таким же як і сформований секретний ключ на сервері.

17) Отримавши секретний ключ серверу ми розшифровуємо передані дані в зашифрованому виді.

18) Конвертуємо отриманий рядок до першопочаткового формату.

Рис. 2.15. Блок-схема запропонованого методу

В ході цього алгоритму ми отримуємо захищену систему передачі, де для кожного запиту всі дані які передаються будуть зашифровані та зашифровані дані не будуть повторюватися навіть якщо тіло запитів біде абсолютно однаковим. Завдяки

використанню алгоритму еліптичної криптографії під час процесу передачі даних у нас утвоюється між сервером та клієнтом сеансова пара, чрезе те що в нас секретний ключ пристрою утвоюється на основі публічного ключу іншого пристрою, секретні ключі обох пристроїв під час одного запросу є однаковими.

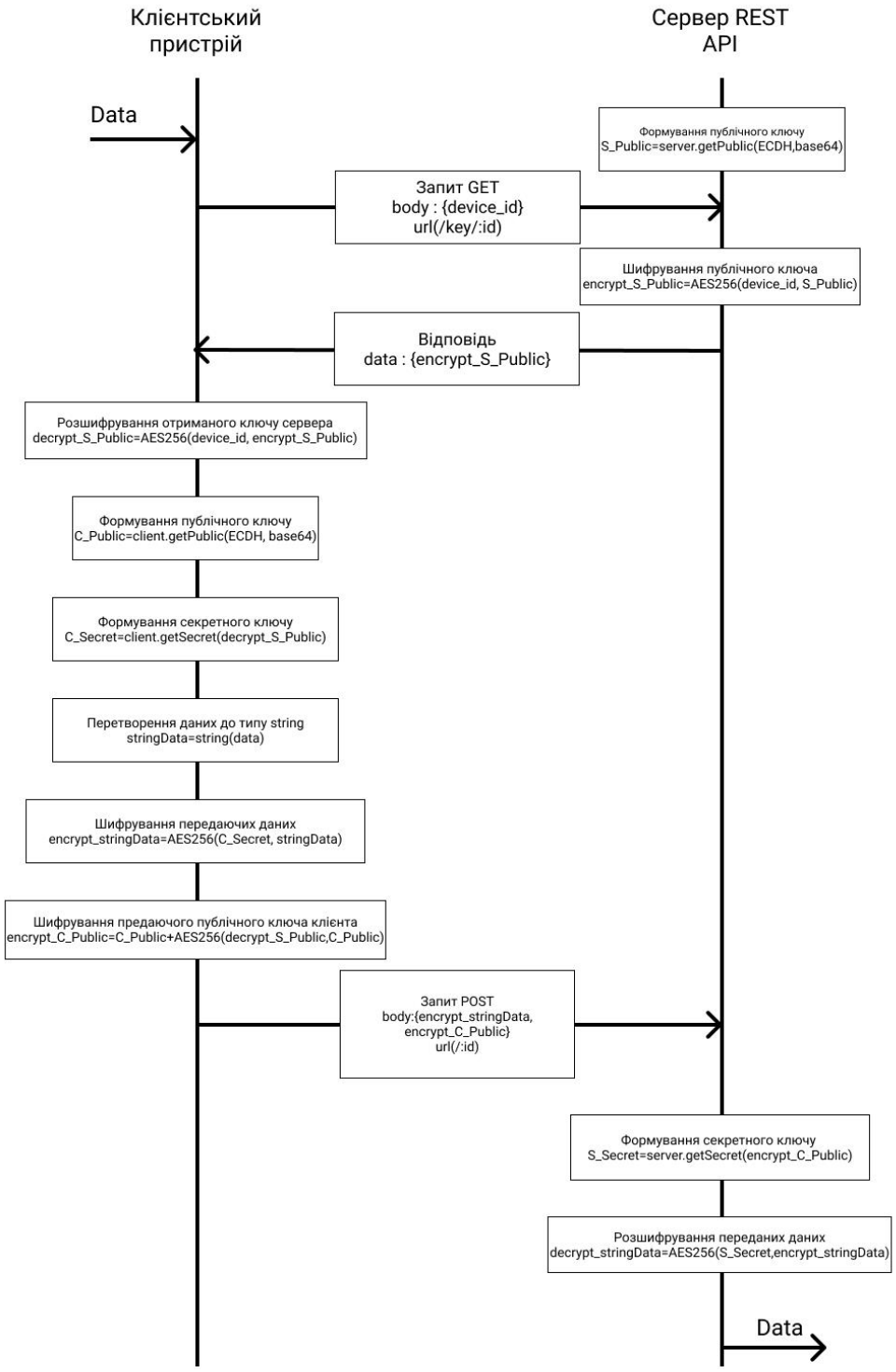

Рис. 2.16. Блок-схема роботи алгоритму запропонованого методу

При виконані запиту на сервер ми виводимо в таблицю дані які ми отримуємо в тілі запиту та розшифровані дані в іншу таблицю.

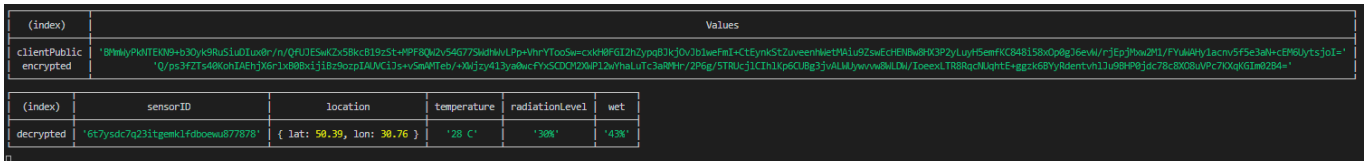

Рис. 2.17. Зображення прийнятих та розшифрованих даних на сервері

Висновки до розділу

В даному розділі проведено огляд та аналіз існуючих рішень підвищення рівню захисту інформації в IoT. Проведено порівняльний аналіз проміжного ПЗ, за допомогою аналізу було обрано прототип для запропонованого методу. Розглянуто використання взятого за прототип метод захисту проміжного ПЗ в мережі інтернету речей, приведено детальну схему використання в мережі інтернету речей, розглянуто переваги та недоліки його використання, обрано, що на основі представлених недоліків буд проводитися модифікування.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ КОНТРОЛЮ АВТОМАТИЧНИХ ОНОВЛЕНЬ В СИСТЕМАХ IoT

3.1 Представлення особливостей роботи системи в режимі адміністратора

Адміністратор представляє вміст інтерфейсу користувача під час завантаження нової версії програми на сервер бази даних. Адміністратор — це інструмент, призначений для контролю оновлення версій програм. Ним може маніпулювати лише системний адміністратор для забезпечення безпеки. Щоб полегшити роботу адміністратора, розробіть «графічний інтерфейс адміністратора», який показано на рис. 3.1. Цей графічний інтерфейс містить дві частини: інтерфейс дисплея та інтерфейс керування.

Інтерфейс дисплея містить піктограму вгорі та назву програми, час випуску, поточну версію. Крім того, він містить примітку внизу, яка використовується для нагадування користувачеві.

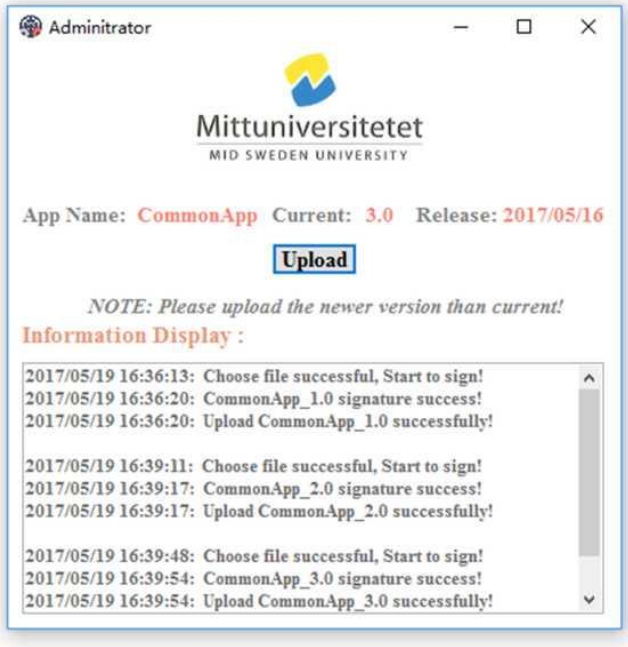

Рис. 3.1. Результат роботи адміністратора

Інтерфейс керування містить кнопку «Завантажити», яка є ключем графічного інтерфейсу користувача, оскільки її можна використовувати для виконання функції завантаження. Коли адміністратор натисне цю кнопку, з'явиться вікно «Вибір файлу».

Як показано на рис. 3.2, якщо вибраний архів існував, з'явиться вікно з попередженням «Файл існував! Будь ласка, виберіть повторно». Якщо вибраний архів старіший за поточний, з'явиться вікно попередження «Версія для завантаження старіша за поточну! Виберіть ще раз!». Якщо із завантаженим архівом немає проблем, з'явиться сповіщення про успішне завантаження.

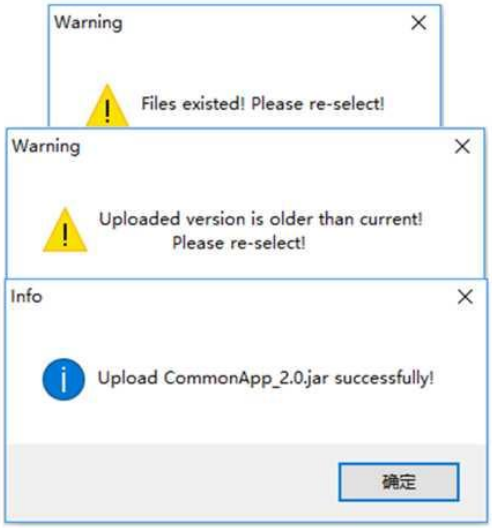

Рис. 3.2. Вікно підказок для різних сценаріїв

Щоб полегшити адміністратору перегляд історії завантажень, додайте функцію журналу в програму адміністратора, як показано на рис. 3.3. Це автоматично записуватиме джерело програми, час випуску, назву програми, номер версії та час завантаження кожного разу, коли адміністратор завантажує файл.

Рис. 3.3. Представлення вмісту log_upload.txt.

Основною функцією сервера Tomcat є надання бази даних, як показано на рис. 3.4. У верхній частині цього малюнка є URL-адреса веб-сайту бази даних. Це " http://localhost:9999/myAuto/Datebase/ ". "localhost" - це ім'я хоста, а 9999 - це номер порту, який можна налаштувати. «myAuto» — це ім’я проекту сервера, а «Database» — підпапка проекту.

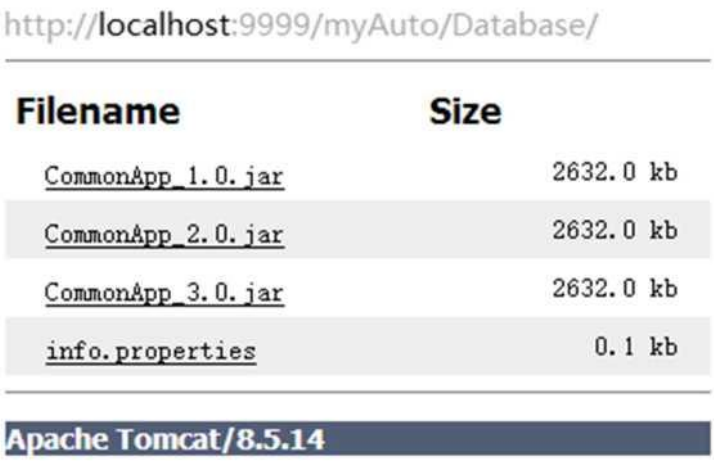

Рис. 3.4. База даних програми на сервері Tomcat

Середина рисунка показує вміст папки бази даних. Ця папка містить файли двох типів: файл JAR і файл властивостей. може існувати кілька файлів JAR, які є файлами програми, які можна виконати, і ці файли сортуються за порядком. Для файлу властивостей завжди існує лише один файл цього типу, який можна вважати файлом конфігурації, оскільки він завжди записує інформацію про поточну найновішу версію файлу JAR. Файл властивостей надзвичайно важливий для менеджера, щоб перевірити найновішу версію файлу JAR.

3.2 Опис основних інтерфейсних елементів системи

Результат менеджера показано на рисунку 3.5. Вміст інтерфейсу користувача розділено на дві категорії згідно: завантажувач і відправник. Наступні два пункти пояснять детальний інтерфейс керування та інтерфейс відображення згідно з рисунком 3.5.

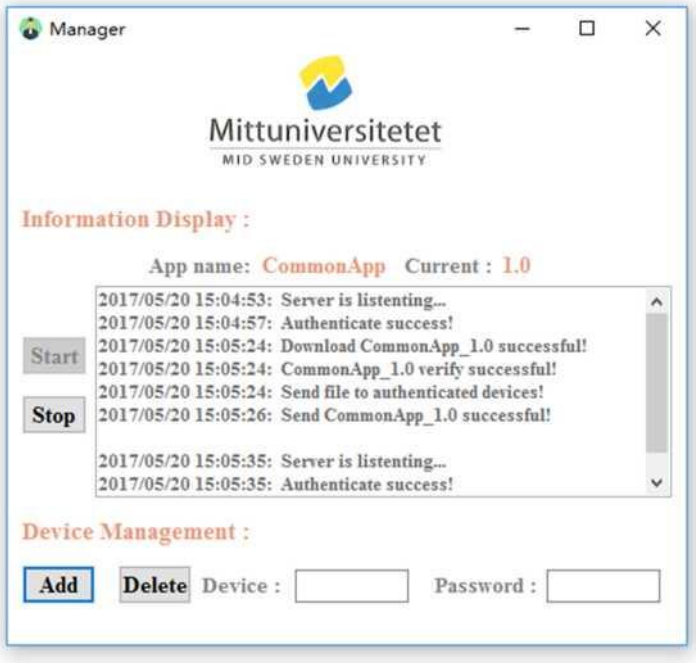

Рис. 3.5. Результат Менеджера

Кнопка «Пуск» може ініціювати запуск завантажувача. Якщо натиснути «Пуск», запуститься таймер завдання завантаження, який регулярно запускатиметься повторно. Якщо менеджер хоче зупинити таймер, натиснувши кнопку «Зупинити». Якщо припустити, що таймер активний і є JAR-файл новішої версії, програма завантаження завантажить його у свою локальну систему та відобразить «Завантаження xxx_xxx успішно!» Потім перевірте цілісність файлу за допомогою відповідного інструменту. Якщо це вдалось, відобразіть «xxx_xxx verify successful» і оновіть «Інформаційний дисплей», який містить «Назва програми» та «Поточний» номер версії. У той же час запишіть файл журналу, як показано на рис. 3.6. Цей файл журналу містить час випуску, назву програми та номер версії файлу JAR і фактичний час завантаження для завантажувача.

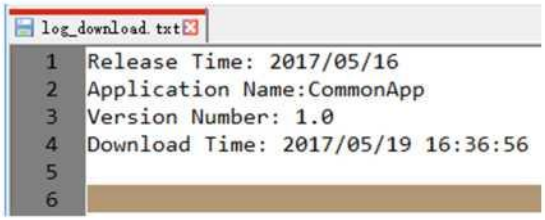

Рис. 3.6. Завантаження вмісту log_download.txt.

Відправник запускається автоматично після запуску програми Manager. Він надрукує «Сервер прослуховує» та чекає, поки пристрій підключиться. Якщо є будь-який пристрій для підключення, то ці дві сторони автентифікуватимуть одна одну. У разі успіху виведіть «Автентифікація успішна!», інакше «Автентифікація не виконана!» .Тільки результат автентифікації є успішним, відправник чекатиме, поки завантажувач видасть команду надсилання. Якщо відправник отримає команду, він надішле файл на пристрій і повідомить про результат надсилання в інтерфейсі користувача.

Частина керування пристроєм у нижній частині інтерфейсу призначена для керування списком довірених пристроїв за допомогою «Додати» та «Видалити» пристрій. Функція «Видалити» вимагає правильного введення імені пристрою та пароля. А функція «Додати» вимагає, щоб введення імені пристрою та пароля не було нульовим.

Результат роботи пристрою розділений на дві частини. Перший – це результати приймача перед установкою. Інший результат інсталятора під час встановлення нової програми.

Процес приймання розпочнеться, доки запуститься програма пристрою. Як показано на рис. 3.7, існує три типи дисплеїв. Верхня частина інтерфейсу призначена для відображення інформації про поточну запущену програму, таку як назва програми та поточна версія програми. Середня частина інтерфейсу користувача використовується для відображення підключення та інформації про завантаження. Коли пристрій успішно з’єднається із сервером, на ньому буде надруковано повідомлення «З’єднатися із сервером успішно». Після підключення Пристрої автентифікуватимуться за допомогою Менеджера один одного, а потім надрукують повідомлення «Автентифікація успішна/не виконана!» відповідно до різного результату.

Водночас у тексті «Інформація про встановлення» внизу інтерфейсу користувача відображатиметься інформація та загальний час роботи поточної запущеної програми. Продовжувати роботу програми є нормальним явищем у

реальному житті, тому час роботи використовується для вказівки на те, що програма виконується, а не виходу.

Дисплей отримання показаний на рис. 3.8. Якщо цей процес продовжується після успішної автентифікації, отримувач завжди чекатиме, доки не надходитиме файл JAR. Потім приймач отримує, зберігає файл і друкує

«Завантажено xxx успішно!» Потім перевірте його, якщо здається, що результат прийнятний, тоді підготуйтеся до встановлення.

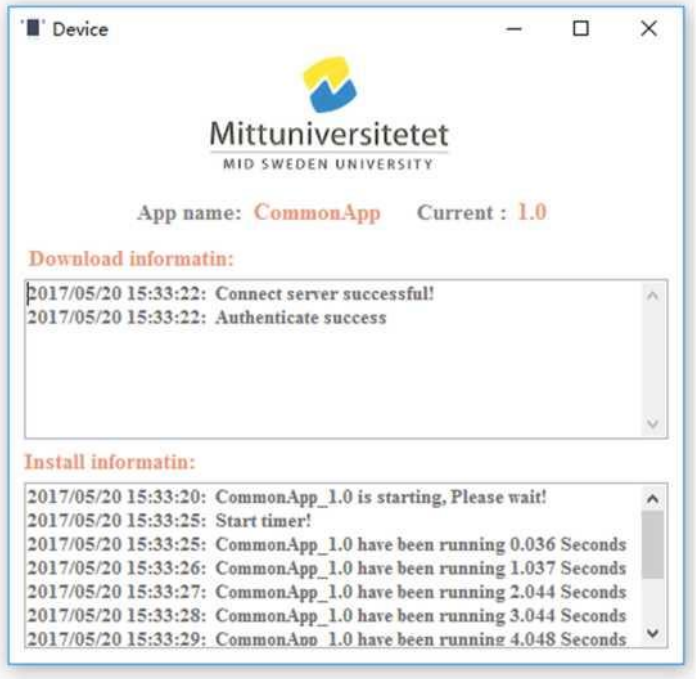

Рис. 3.7. Результат приймача пристрою

Інсталятор використовується для встановлення нової програми. Результат інсталяції показано в тексті «Інформація про інсталяцію» на рис. 3.8. Коли пристрій починає інсталювати програму, «Інформація про інсталяцію» вказуватиме «Підготовка до інсталяції xxx_xx». І роздрукує інформацію про програму, таку як ім’я, постачальник, розташування, ліцензія і т. д. Потім приймач завантажить JAR-файл і почне справжнє встановлення.Щоб краще спостерігати за процесом встановлення, він виведе прогрес встановлення.

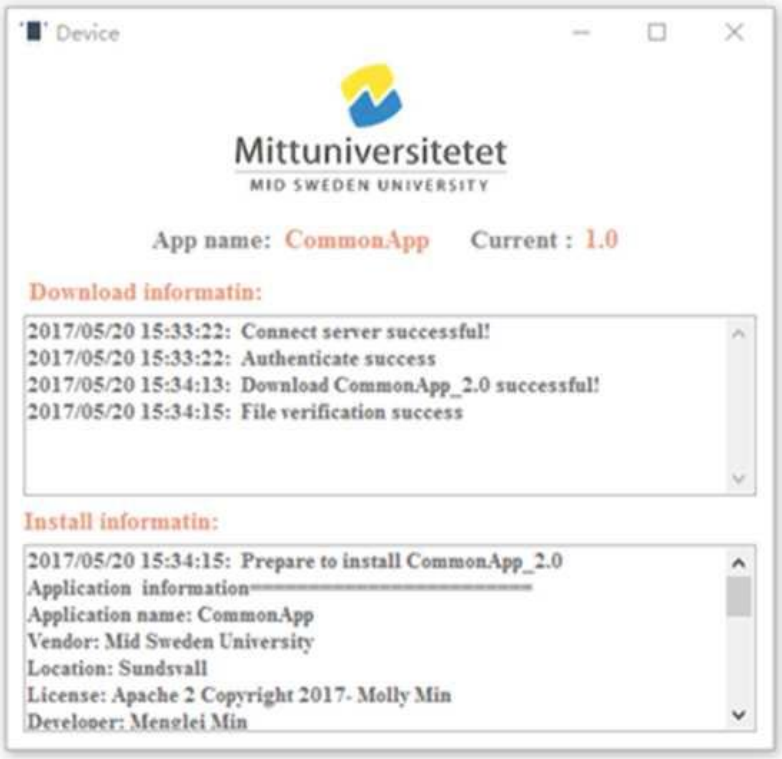

Рис. 3.8. Результат інсталяції пристрою

Коли інсталяція завершиться, процес видасть «Install End». І зареєструйте інформацію про інсталяцію певної програми, як показано на рис. 3.9. У цьому файлі буде записано назву програми, версію та час інсталяції. Одночасно виконайте команду «Restart». в процесі. Коли процес виконує інструкцію «Перезапустити», весь вміст інтерфейсу користувача буде очищено, а Пристрій буде перезапущено. Після перезапуску Пристрій запустить програму найновішої версії.

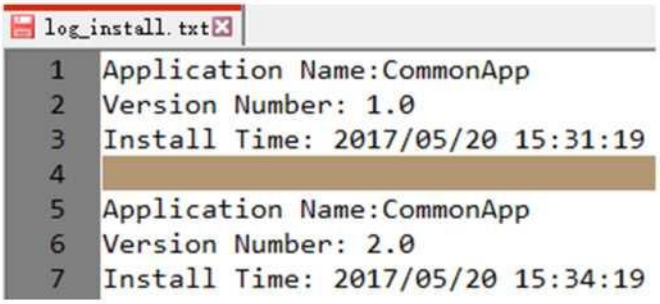

Рис. 3.9. Вміст log_instaH.txt.

3.3 Оцінка та аналіз отриманих результатів дослідження

Цей розділ містить вимірювання та оцінку продуктивності та безпеки. Метою цього розділу є оцінка продуктивності та рівня безпеки цієї системи за допомогою деяких технічних і математичних методів. Обсяг оцінки продуктивності збирається на затримці. А оцінка безпеки фокусується на результатах підпису, верифікації та автентифікації.

Для автоматизованої системи оновлення часто важлива своєчасність оновлення системи. Якщо розробник випустив нову версію через серйозну помилку старої версії, але через величезну затримку пристрій не встановив нову версію протягом певного періоду часу, що може спричинити серйозні наслідки. Отже, перш ніж система буде офіційно випущена, тестування різних затримок є дуже необхідним, що зрештою визначить, чи можна запустити систему автоматичного оновлення в реальному житті.

У цьому проекті є різні затримки, але порахувати всю затримку нереально. Тому виберіть декілька затримок, які мають більший вплив на продуктивність системи. Як показано в наведеній нижче таблиці 3.1, завантаження, перевірка менеджера, передача, перевірка пристрою, встановлення, перезапуск і фактична загальна затримка перераховані для вимірювання. Тип 1, 2, 3, 4, 5, 6 є важливою проміжною затримкою в системі автоматичного оновлення. Тип затримки 7 – це фактична загальна затримка від успішного завантаження JAR адміністратором до перезавантаження пристрою. Фактична загальна затримка є найважливішою затримкою, яка безпосередньо стосується продуктивності системи. Для оцінки ефективності необхідно досягти трьох цілей:

Ціль 1: знайти проміжну затримку, яка найбільше відповідає фактичній затримці.

Ціль 2: знайти найбільш пов’язану причину коливань фактичної загальної затримки.

Ціль 3: Оцінити ефективність цієї системи автоматичного оновлення відповідно до результатів вимірювання.

Варто відзначити, що таймер перевірки Менеджера встановлено на 10 секунд. Тому менеджер завжди перевірятиме базу даних кожні десять секунд. Цей таймер важливий для результату вимірювання. Виміряйте ці типи затримки відповідно до методу, наведеного в таблиці 3.1, крім того, додайте рядок під назвою total, який є загальною кількістю затримок для шести проміжних затримок.

Таблиця 3.1.

Представлення результатів

Num.	Start time	End time	Latency types
1	Administrator upload successfully	Manager download successful	Download
2	Manager verify start	Manager verify end	Manager verification
3	Manager sends JAR successfully	Device receive JAR successfully	Transmission
4	Device verify start	Device verify end	Device verification
5	Device installation starts	Device installation ends	Installation
6	Device restart starts	Device restart over	Restart
7	Administrator upload successfully	Device restart over	Actual total

А потім обчисліть їхню середню затримку та стандартне відхилення для кожного типу затримки, результати наведено в таблиці 3.2.

Легко помітити, що затримка передачі дорівнює 0, а перевірка менеджера та затримка перезапуску не змінюються весь час. Отже, стандартне відхилення цих трьох типів затримки дорівнює 0, що є приголомшливою стабільністю. Для верифікації пристрою та затримки встановлення стандартне відхилення дещо змінюється, вони залишаються відносно стабільними. Однак стандартне відхилення

для завантаження, загальної та фактичної загальної затримки значно змінюється та виглядає дуже нестабільним. Таким чином, надійність можна розділити на три категорії відповідно до стандартного відхилення: стабільна, відносно стабільна, нестабільна.

Таблиця 3.2.

Результати вимірювання за типом затримки

Number / Latency/s	1st	2nd	3rd	4th	5th	6th	Average	Stdev
Download	7	3	8	1	1	7	4.5	3.209361
Manager verification	2	2	2	2	2	2	2	0
Transmission	0	0	0	0	0	0	0	0
Device verification	2	2	1	2	2	1	1.666667	0.516398
Installation	4	4	5	4	4	4	4.166667	0.408248
Restart	5	5	5	5	5	5	5	0
Total	20	16	21	14	14	19	17.33333	3.076795
Actual Total	21	17	22	14	14	20	18	3.521363

Щоб краще спостерігати за результатами вимірювань, візуалізуйте результати таблиці 3.2 на лінійній діаграмі, як на рис. 3.1. З цього результату візуалізації легко помітити, що тенденція завантаження, загальної та фактичної затримки однакова. Отже, є підстави вважати, що між цими трьома затримками існує величезний зв’язок. Більше того, загальна затримка — це лише арифметичні результати шести проміжних затримок, тому лише затримку завантаження можна вважати найбільш релевантною проміжною затримкою. І в основному можна сказати, що між іншими затримками немає зв’язку.

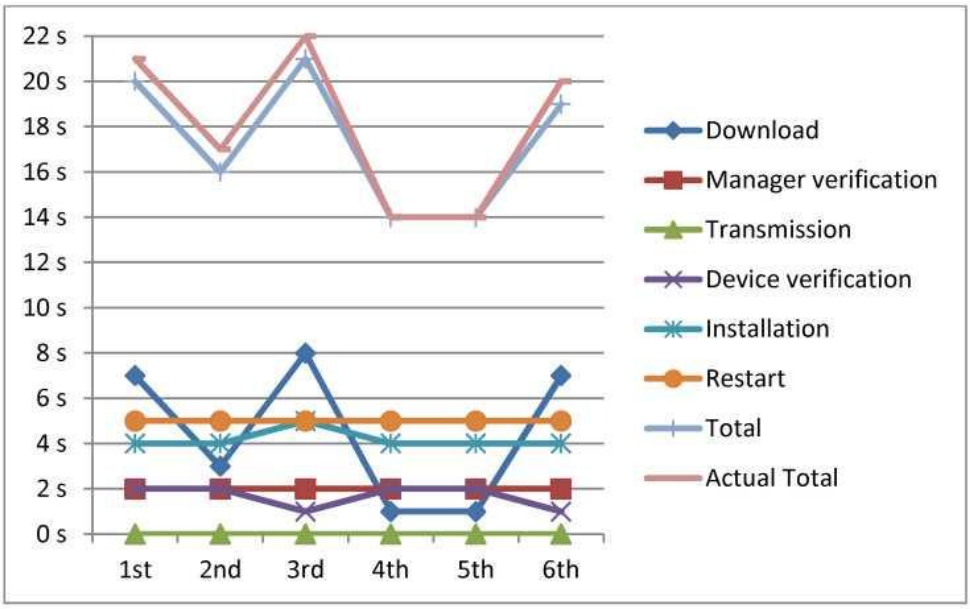

Рис. 3.10. Візуалізація результатів

Для цілі 1 відповідь можна знайти з результату візуалізації, це затримка завантаження.

Для цілі 2 можна зробити такий висновок, що затримка завантаження спричиняє величезні коливання фактичної загальної затримки. Це тому, що таймер перевірки Менеджера становить 10 секунд. Отже, коли файл JAR завантажено, таймер може бути в режимі сну або збирається запуститися. Це призведе до того, що фактичний час сильно відрізнятиметься.

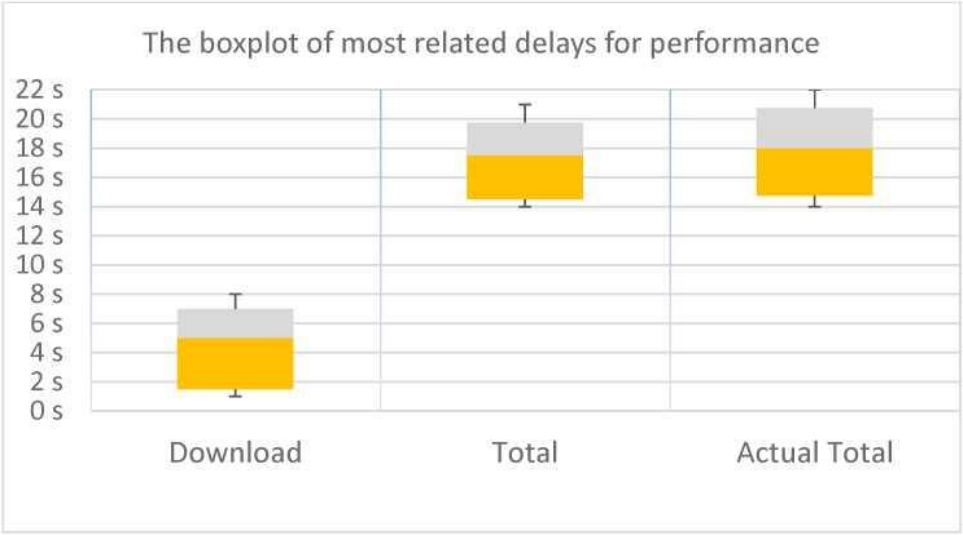

Рис. 3.11. Графік більшості пов’язаних затримок продуктивності

Для цілі 3 середня фактична загальна затримка становить 18 секунд, це хороший результат на основі 10-секундного таймера. Нарешті, цю систему можна розглядати як систему з низькими затримками, певною мірою продуктивність хороша.

Як видно з рисунка 3.10, проміжною затримкою, найбільш пов'язаною із загальною затримкою, є затримка завантаження. Щоб краще показати затримку флуктуацій, зробіть коробковий графік на рис. 3.11 завантаження, загального та фактичного загального.

3.4 Опис процесів забезпечення безпеки системи

Для системи немає сенсу повністю реалізувати очікувану функціональність, але це не гарантує безпеки. Таким чином, мета цього проекту на додаток до функції автоматичного оновлення, безпека також є важливим фактором в одному з міркувань. Тобто основна функція цієї системи автоматичного оновлення розділена на дві частини: безпека та автоматичне оновлення. У цьому розділі оцінюється безпека системи, щоб побачити, чи відповідає вона поставленій меті.

Ця система автоматичного оновлення застосовує три технології для забезпечення безпеки: DSA, SSA та SHA. У цій системі кожна технологія не має абсолютно однакового фокусу, який наведено в таблиці 3.3.

Таблиця 3.3.

Опис елементів безпеки в системі

Object	Technology	Stage	Guarantee
JAR	Digital signature algorithm	Verification	Integrity, Non-repudiation
Manager & Device	Secure sockets layer	Connection	Authenticity, Confidentiality
Manager & Device	Secure hash algorithm	Authentication	Authenticity

Оцінка зосереджена на результатах верифікації, підключення та аутентифікації. На етапі перевірки ця система досягла цілісності та незаперечності, щоб переконатися, що встановлений JAR не пошкоджений і джерело JAR вірне. На етапі з'єднання ця система гарантує автентичність і конфіденційність, щоб гарантувати, що інша сторона є власним довіреним об'єктом, а весь переданий вміст буде зашифровано ключами сеансу. На етапі автентифікації система забезпечувала автентичність через список довірених пристроїв менеджера.

Виміряйте результат кожного етапу з різними попередніми умовами та обчисліть їхній загальний показник успіху за таблицею 3.4. Для етапу перевірки п'ять разів пошкоджений JAR і десять разів пошкоджений JAR до комп'ютера відсоток успішності. Для етапу підключення змініть попередню умову на правильний менеджер і пристрій і неправильний менеджер або пристрій. Для етапу автентифікації це те саме, що етап підключення.

Таблиця 3.4.

Рівень успішності кожного етапу з різними попередніми умовами

Stage	Object	Frequency	Success rate
Verification	Uncorrupted JAR	5	100%
	Corrupted JAR	10	0%
Connection	Right Manager & Device	5	100%
	Wrong Manager or Device	10	0%
Authentication	Right Manager & Device	5	100%
	Wrong Manager or Device	10	0%

Рівень успішності правильної попередньої умови становить 100%, а в інших випадках – 0%, що є приємним результатом. Певною мірою систему можна вважати достатньо безпечною, оскільки в ній досягнуто цілісності, незаперечності,

автентичності та конфіденційності. Однак система може бути скомпрометована за деяких екстремальних умов, наприклад підробки сертифіката, оскільки система не може ідентифікувати цей підроблений сертифікат, який відображається з правильним.

Висновки до розділу

В даному розділі виконано реалізацію інформаційної технології контролю автоматичних оновлень в системах IoT. Виконано представлення особливостей роботи системи в режимі адміністратора, описано основні інтерфейсні елементів системи, проведено оцінку та аналіз отриманих результатів дослідження.

ВИСНОВКИ

В магістерській роботі досліджено моделі, методи та засоби контролю автоматичних оновлень в системах IoT. Загальним результатом цього проекту є оцінка та впровадження безпечної системи автоматичного оновлення на основі Інтернету речей. Конкретні цілі цього проекту підтверджуються наступним чином:

Згідно цілі 1, необхідно було знайти три автоматичні оновлення існуючих робіт, що було досягнуто. Ці три пов'язані елементи це «Windows Update», «Java Web Start» і «Getdown» відповідно.

Також була досягнута ціль 2, тобто знайти три рішення для забезпечення безпеки. Ці три безпечні методи були пояснені в роботі, а саме це безпечний хеш-алгоритм, цифровий підпис і рівень безпечних сокетів. Також було продемонстровано їх достатню безпеку для захисту системи.

Ціль 3 магістерської роботи також було досягнуто. Відповідно запропоновано загальну мережеву архітектуру відповідно до поєднання трьох існуючих пов'язаних елементів і трьох безпечних методів. Адміністратор і пристрій завжди з'єднуються та автентифікуються один з одним, оскільки в той же час менеджер перевіряє базу даних кожен період. Коли адміністратор завантажує програму в базу даних, менеджер завантажує та надсилає її на пристрій. Якщо все пройде гладко, пристрій встановить програму та зрештою перезапуститься.

Ціль 4, стосується сценарій передбачається в промисловому Інтернеті речей. Система була реалізована з використанням мови програмування Java і може успішно виконувати безпечні автоматичні оновлення програми Java. Технічні підходи до реалізації були детально описані.

Ціль 5 полягає в оцінці продуктивності і безпеки. Перевагами продуктивності є фактична загальна затримка та проміжна затримка. Виміряні результати показали, що система автоматичного оновлення може бути завершена протягом двадцяти секунд, що є коротким періодом, тому систему можна вважати високопродуктивною. Що стосується безпеки, результати вимірювання успішності

системи автоматичного оновлення трьох різних етапів різних об’єктів доводять, що система автоматичного оновлення достатньо безпечна.

Отже, автоматичне оновлення для промислового Інтернету речей — це те ж саме, що й Інтернет речей, що принесе етичні проблеми. Він має три основні наступні проблеми:

По-перше, збільшити рівень безробіття, тому що оновлення автоматизації замінить оновлення вручну, що означає заміну штату цієї посади.

По-друге, стабільність системи знижується, як тільки вразливості системи використовуються, масштаб шкоди буде охоплювати всі пристрої заводу, і навіть призводить до повного паралічу заводу.

По-третє, це питання відповідальності. Хто повинен нести відповідальність за поганий результат оновлення, зроблений цією системою автоматичного оновлення.

Для файлу оновлення ця система призначена для одного файлу JAR, але більшість інсталяційних файлів мають кілька різних форм. Отже, якщо буде більше часу, цю систему можна розширити з одного файлу JAR на кілька пов’язаних файлів. Для продуктивності цієї системи складно оцінити доступність і масштабованість. Дійсно, ці дві переваги також були важливими для промислової системи. Що стосується доступності, можливо, у майбутньому можна буде розглянути послугу резервного копіювання. Що стосується масштабованості, слід розробити більше імітованих пристроїв і спробувати підключитися до цієї системи одночасно.

З точки зору безпеки, недоліком цієї системи є те, що вона не може розпізнати зловмисний спосіб, якщо хтось володіє легальним сертифікатом. Якщо цю систему атакувати таким способом, зловмисник може викрасти дані, не знаходячись. Тож цей тип атаки слід враховувати та посилювати відповідну політику безпеки.

Запропоновано модифікований метод захисту інформації в мережі інтернету речей, який базується на REST API та на методі еліптично-кривої криптографії. Для даного методу приведено алгоритм та блок схему роботи даного методу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Evans D. The internet of things: How the next evolution of the internet is changing everything[J]. CISCO white paper, 2011, 1(2011):1-11.
2. Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." Computer networks 54.15 (2010): 2787-2805.
3. KrebsOnSecurity, “This is Why People Fear the ‘Internet of Things’ ”, https://krebsonsecurity.com/2016/02/this-is-why-peoplefear-the-internet-of-things/Published 2016-02.
4. Xia F, Yang L T, Wang L, et al. Internet of things[J]. International Journal of Communication Systems, 2012, 25(9): 1101.
5. Keertikumar M, Shubham M, Banakar R M. Evolution of IoT in smart vehicles: An overview[C]//Green Computing and Internet of Things (ICGCIoT), 2015 International Conference on. IEEE, 2015: 804-809.
6. Hou B, Sheng-Yang Y Y M Q. Design of Distributed Remote Real- Time Monitoring and Control System Based on Internet[J]. Journal of Emerging Trends in Computing and Information Sciences, 2012, 3(7).
7. Internet of Things,https://en.wikipedia.org/wiki/Internet_of_things
8. Want R. An introduction to RFID technology[J]. IEEE pervasive computing, 2006, 5(1): 25-33.
9. RFID, https://www.barcodesinc.com/info/buying-guides/rfid.htm.
10. Secure Hash Algorithm, https://en.wikipedia.org/wiki/Secure_Hash_Algorithms
11. Digital signature, https://en.wikipedia.org/wiki/Digital_signature
12. Bhiogade M S. Secure socket layer[C]//Computer Science and Information Technology Education Conference. 2002: 85-90.
13. Secure Sockets Layer, https://www.ibm.com/developerworks/cn/java/j-lo-ssltls/
14. Stoyle, Gareth. "A theory of dynamic software updates." (2007).
15. Windows Update, http://www.blogtechnika.com/what-iswindows-update-and-how-it-works
16. Java Web Start, https://www.java.com/zh_CN/download/faq/java_webstart.xml,

17. Zukowski J. Deploying software with jnlp and java web start[J]. Technical Article. August, 2002.
18. Getdown, https://github.com/threerings/getdown/, Received
19. Weber R H. Internet of Things–New security and privacy challenges[J]. Computer law & security review, 2010, 26(1): 23-30.
20. Vlacheas P, Giaffreda R, Stavroulaki V, et al. Enabling smart cities through a cognitive management framework for the internet of things[J]. IEEE Communications Magazine, 2013, 51(6): 102-111.
21. Choi B C, Lee S H, Na J C, et al. Secure firmware validation and update for consumer devices in home networking[J]. IEEE Transactions on Consumer Electronics, 2016, 62(1): 39-44.
22. Maven Shade Plugin, https://maven.apache.org/plugins/mavenshade-plugin/
23. Apache Tomcat, https://en.wikipedia.org/wiki/Apache_Tomcat.
24. Rescorla E. SSL and TLS: designing and building secure systems [M]. Reading: Addison-Wesley, 2001.
25. The Internet of Things: A Critique of Ambient Technology and the All-Seeing Network of RFID [Електронний ресурс] – Режим доступу до ресурсу: https://www.networkcultures.org/_uploads/notebook2_theinternetofthings.pdf.
26. Наукове дослідження Інтернету речей Hewlett Packard Enterprise [Електронний ресурс] – Режим доступу до ресурсу: https://json.tv/tech_trend_find/nauchnoe-issledovanie-interneta-veschey-ot-ewlett-packard-enterprise-20160503115845. 6. Xiang Y. Low-rate DDoS attacks detection and traceback by using new information metrics / Y. Xiang, K. Li, W. Zhou. // IEEE Transactions on Information Forensics and Security. – 2011. – С. 426–437.
27. Що таке Шодан? [Електронний ресурс] – Режим доступу до ресурсу: https://help.shodan.io/the-basics/what-is-shodan.
28. Shodan: The scariest search engine on the Internet [Електронний ресурс] – Режим доступу до ресурсу: https://money.cnn.com/2013/04/08/technology/security/shodan/index.html.

29. Dahua IP Camera 3.200.0001.6 access control [Електронний ресурс] – Режим доступу до ресурсу: https://vuldb.com/?id.99110.
30. John the Ripper [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/John_the_Ripper.
31. Byres J. Honeywell selects TofinoTM Modbus Read-only Firewall to Secure Critical Safety Systems. [Электронный ресурс]. The University of British Columbia. Canada. 2011. January 6. URL: http://www.tofinosecurity.com/sites/default/files/pr_hon_modbus_read-only_firewall_01_06_11.pdf
32. The Untouchables: Protecting Sensitive Technology Systems with Tenable's Passive Vulnerability Scanner // Tenable Network Security. [Электронный ресурс]. Tenable Network Security, Inc. Columbia, USA. 2012. January 9. URL:http://static.tenable.com/whitepapers/Tenable-TheUntouchables.pdf
33. Securing IoT Devices and SecurelyConnecting the Dots Using REST API and Middleware [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/8777334.
34. Система аутентифікації на базі еліптичних кривих з використанням векторних операцій [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://ela.kpi.ua/bitstream/123456789/25522/1/Albrekht_magistr.pdf.
35. Implementation and integration of efficient ECDH key exchanging mechanism in software based VoIP network [Електронний ресурс] // IEEE. – 2011. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/5972416.
36. Scully P. IoT Security Architecture on the Device and Communication Layers [Електронний ресурс] / Padraig Scully. – 2019. – Режим доступу до ресурсу: https://dzone.com/articles/iot-security-part-1-of-3-architecture-on-the-device-and-communication-layers.
37. Nandi A. An Overview: Security Issue in IoT Network [Електронний ресурс] / A. Nandi, M. Agarwal, D. Samanta // IEEE. – 2018. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/8653728.
38. Lightweight Data Security Model for IoT Applications: A Dynamic Key Approach [Електронний ресурс] / M. Kumar, S. Kumar, R. Budhiraja, S. Singh // IEEE. –

2019. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/7917128.

39. Tewari, A., & Gupta, B. B. (2018, January). A robust anonymity preserving authentication protocol for IoT devices. In Consumer Electronics (ICCE), 2018 IEEE International Conference on (pp. 1-5). IEEE
40. Wu, F., Xu, L., Kumari, S., & Li, X. (2017). “A privacy-preserving and provable user authentication scheme for wireless sensor networks based on internet of things security”. Journal of Ambient Intelligence and Humanized Computing, 8(1), 101-116.
41. Hsieh, W. B., & Leu, J. S. (2014). A Robust ser Authentication Scheme sing Dynamic Identity in Wireless Sensor Networks. Wireless personal communications, 77(2), 979-989.
42. Chien, H. Y. (2007). SASI: A new ultralightweight RFID authentication protocol providing strong authentication and strong integrity. IEEE transactions on dependable and secure computing, 4(4), 337-340.