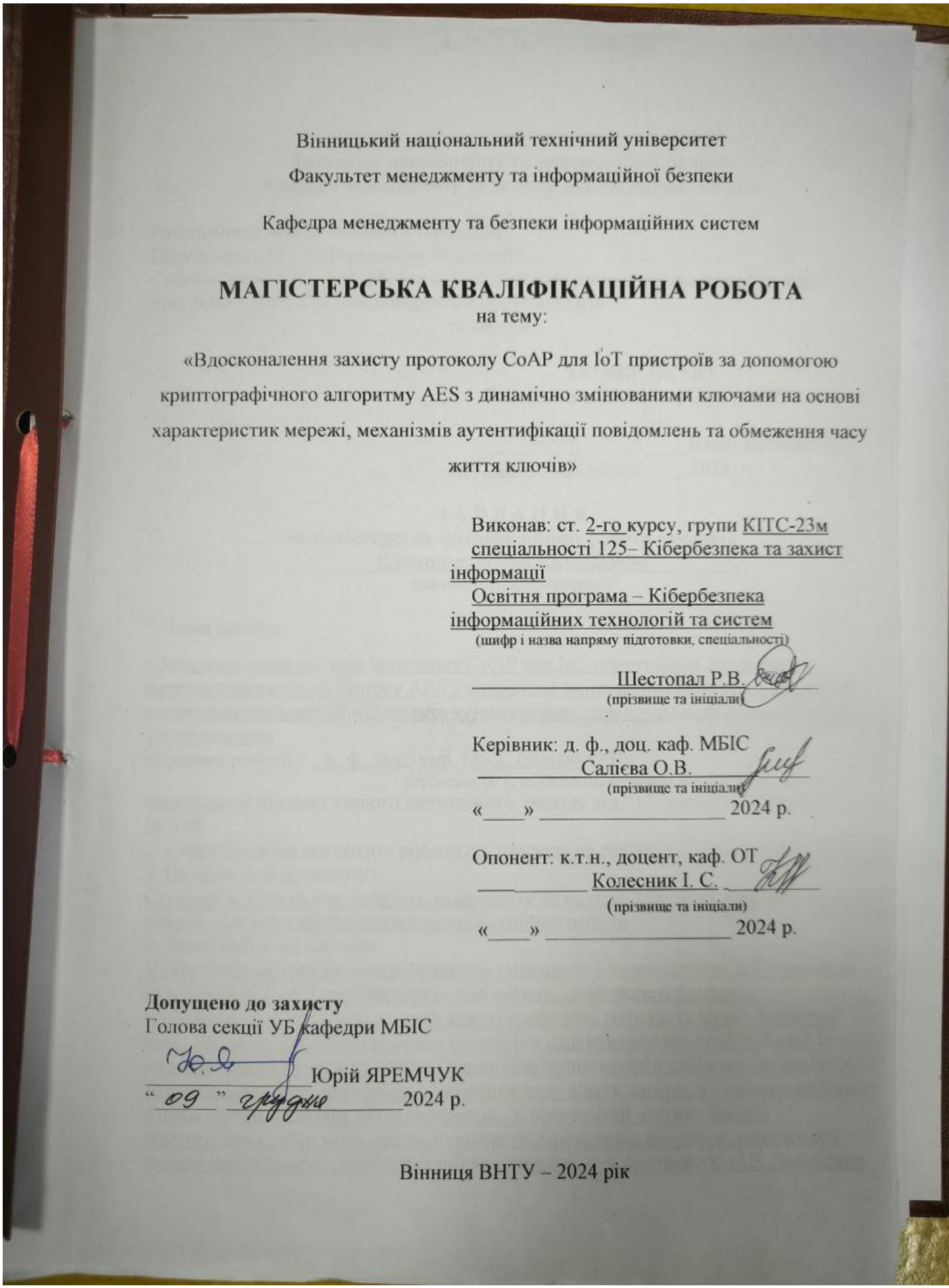


Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Вдосконалення захисту протоколу CoAP для IoT пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів»

Виконав: ст. 2-го курсу, групи КІТС-23м
спеціальності 125– Кібербезпека та захист інформації
Освітня програма – Кібербезпека інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

_____ Шестопал Р.В.
(прізвище та ініціали)

Керівник: д. ф., доц. каф. МБІС
_____ Салієва О.В.
(прізвище та ініціали)
«____» _____ 2024 р.

Опонент: к.т.н., доцент, каф. ОТ
_____ Колесник І. С.
(прізвище та ініціали)
«____» _____ 2024 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
_____ Юрій ЯРЕМЧУК
“09” грудня 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека та захист інформаціїОсвітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС
_____ **Юрій ЯРЕМЧУК**
"20" вересня 2024 р.

З А В Д А Н Н Я
на магістерську кваліфікаційну роботу студенту
Шестопал Роман Валерійович
(прізвище, ім'я, по-батькові)

1. Тема роботи:

«Вдосконалення захисту протоколу CoAP для IoT пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів»
Керівник роботи : д. ф., доц. каф. МБІС Салієва О.В.
(прізвище, ім'я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "17" вересня 2024 року № 310
2. Строк подання студентом роботи за тиждень до захисту.
3. Вихідні дані до роботи:
Стандарти, електронні джерела, підручники та наукові статті по темі, які стосуються теми магістерської кваліфікаційної роботи.
4. Зміст текстової частини:
У текстовій частині роботи розглянуто теоретичні основи захисту IoT-пристроїв та протоколу CoAP, включаючи їх важливість, особливості безпеки, характеристику протоколу, огляд криптографічних методів та аналіз існуючих підходів до захисту. Далі описано створення вдосконаленого захисту CoAP із використанням алгоритму AES із динамічно змінюваними ключами, механізмів аутентифікації повідомлень та обмеження часу життя ключів, а також розроблено алгоритм взаємодії між IoT-пристроями. У програмній частині роботи обґрунтовано вибір мови програмування та середовища розробки, реалізовано модулі криптографії, аутентифікації та інтеграції їх у протокол CoAP. Проведено

тестування реалізованого протоколу, що підтвердило його ефективність, та сформульовано висновки щодо отриманих результатів.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)

У дугому розділі магістерської кваліфікаційної роботи наведено 1 рисунок, у третьому розділі – 1 рисунок

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина			
I	Салієва О.В.. д. ф., доц. каф. МБІС		
II	Салієва О.В.. д. ф., доц. каф. МБІС		
III	Салієва О.В.. д. ф., доц. каф. МБІС		
Економічна частина			
IV	Бурєннікова Н. В. д.е.н., професор каф. ЕПВМ		

7. Дата видачі завдання 20 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
1.	Визначення напрямку магістерської роботи, формулювання теми	20.09.2024	31.09.2024	
2.	Аналіз предметної області обраної теми	01.10.2024	15.10.2024	
3.	Розробка роботи	16.10.2024	26.10.2024	
4.	Написання магістерської роботи на основі розробленої теми	27.10.2024	15.11.2024	
5.	Передзахист магістерської кваліфікаційної роботи	16.11.2024	24.11.2024	
6.	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	27.11.2024	04.12.2024	
7.	Захист магістерської кваліфікаційної роботи	10.12.2024	17.12.2024	

Студент (підпис) Шестопал Р.В.

Керівник роботи (підпис) Салієва О.В.

АНОТАЦІЯ

УДК 004.56.5

Шестопал Р.В. Магістерська кваліфікаційна робота зі спеціальності 125 – «Кібербезпека та захист інформації», освітня програма «Кібербезпека інформаційних технологій та систем». Вінниця: ВНТУ, 2024. – 116 с.

Укр. мовою. Бібліогр.: 41 назв; рис.: 2; табл. 10.

Ключові слова: кібезбезпека, IoT, CoAP

Магістерська робота присвячена вдосконаленню захисту протоколу CoAP для IoT-пристроїв шляхом впровадження криптографічного алгоритму AES із динамічно змінюваними ключами, механізмів аутентифікації повідомлень та обмеження часу життя ключів.

.Для забезпечення безперервної безпеки реалізовано механізм динамічного управління ключами, що базується на характеристиках мережі, таких як час дії ключа та кількість сесій. Також розроблено функціонал для автоматичного оновлення ключів, який мінімізує ризик їх компрометації.

Реалізовано модулі шифрування, аутентифікації та їх інтеграцію в протокол CoAP, що забезпечило захищений обмін даними між пристроями. Проведено тестування реалізованого протоколу, яке підтвердило його ефективність, коректність роботи, стійкість до атак на повторення та відповідність вимогам до продуктивності.

Результати дослідження демонструють, що запропонований підхід забезпечує високий рівень безпеки при мінімальних витратах на ресурси, що робить його придатним для використання у великих і динамічних IoT-мережах.

ABSTRACT

UDC 004.56.5

Shestopal R.V. Master's qualification work in specialty 125 - “Cybersecurity and Information Protection”, educational program “Cybersecurity of Information Technologies and Systems”. Vinnytsia: VNTU, 2024. - 116 p.

In Ukrainian. Bibliography: 41 titles; Figures: 2; Table 10.

Keywords: cybersecurity, IoT, CoAP

The master's thesis is devoted to improving the security of the CoAP protocol for IoT devices by implementing the AES cryptographic algorithm with dynamically changing keys, message authentication mechanisms, and key lifetime limitation.

To ensure continuous security, a dynamic key management mechanism based on network characteristics such as key lifetime and number of sessions is implemented. Functionality for automatically updating keys has also been developed, which minimizes the risk of compromise.

Encryption and authentication modules were implemented and integrated into the CoAP protocol, which ensured secure data exchange between devices. The implemented protocol was tested, which confirmed its effectiveness, correct operation, resistance to replay attacks, and compliance with performance requirements.

The results of the study demonstrate that the proposed approach provides a high level of security with minimal resource costs, which makes it suitable for use in large and dynamic IoT networks.

ЗМІСТ

ВСТУП

Сучасний розвиток Інтернету речей (IoT) відкриває нові можливості для автоматизації, моніторингу та управління системами в різних галузях: від промисловості до повсякденного життя. Однак із поширенням IoT-застосунків зростають і ризики, пов’язані з безпекою таких систем. Передача даних між пристроями, які часто мають обмежені обчислювальні ресурси, робить їх вразливими до атак, таких як підробка повідомлень, атаки на повторення чи компрометація ключів шифрування. У зв’язку з цим забезпечення конфіденційності, автентичності та цілісності даних стає першочерговим завданням для побудови безпечних IoT-мереж.

Актуальність. Протокол CoAP (Constrained Application Protocol), який активно використовується в IoT-середовищах, завдяки своїй легковажній архітектурі є вразливим до багатьох видів атак. Стандартні методи захисту не завжди адаптовані до обмежень IoT-пристроїв, що створює потребу в удосконаленні криптографічних механізмів захисту. Використання алгоритму AES із динамічно змінюваними ключами, механізмами аутентифікації повідомлень та обмеження часу життя ключів може забезпечити гнучкість і ефективність захисту, враховуючи специфіку IoT-мереж.

Мета і задачі дослідження. Метою дослідження є вдосконалення захисту протоколу CoAP для IoT-пристроїв шляхом впровадження криптографічного алгоритму AES із динамічно змінюваними ключами, механізмами аутентифікації повідомлень та обмеження часу життя ключів. Для досягнення цієї мети вирішуються такі задачі:

- аналіз існуючих методів захисту протоколу CoAP та особливостей безпеки IoT-систем;
- розробка вдосконаленого алгоритму захисту із використанням AES у режимі GCM;
- реалізація механізмів динамічної зміни ключів та управління часом їхнього життя;

- інтеграція розробленого модуля із протоколом CoAP;

- проведення тестування реалізованого рішення для оцінки його ефективності та продуктивності.

Об’єкт дослідження – IoT-системи, які використовують протокол CoAP для обміну даними.

Предмет дослідження – методи захисту переданих даних у протоколі CoAP із застосуванням криптографічного алгоритму AES із динамічно змінюваними ключами.

Новизна роботи. Запропоновано вдосконалення захисту протоколу CoAP шляхом динамічної адаптації криптографічних ключів до змінних характеристик мережі. Це рішення забезпечує підвищену стійкість до атак на компрометацію ключів, повторення повідомлень та підробку даних, що є новим підходом у захисті легковажних протоколів для IoT.

Практична цінність. Результати дослідження можуть бути використані для забезпечення безпеки IoT-мереж у реальних умовах. Розроблений модуль захисту може бути інтегрований у програмні рішення для IoT-систем, що дозволить забезпечити високий рівень безпеки при мінімальних витратах на ресурси, що особливо актуально для пристроїв із обмеженими обчислювальними можливостями.

Запропонований підхід сприяє підвищенню рівня довіри до IoT-технологій, забезпечуючи захищений обмін даними у динамічних середовищах.

Апробація: тези доповідей представленні на Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» на тему «Аналіз методів захисту IoT-мереж» [12].

1 ТЕОРЕТИЧНІ ЗАСАДИ ЗАХИСТУ IoT ПРИСТРОЇВ ТА ПРОТОКОЛУ CoAP

У цьому розділі буде розглянуто теоретичні основи захисту IoT-пристроїв та протоколу CoAP, який широко використовується у сучасних мережах. Спочатку буде досліджено важливість забезпечення безпеки IoT-систем та проаналізовано специфічні особливості, які впливають на їхню захищеність. Далі буде розглянуто загальну характеристику протоколу CoAP, його роль у мережах IoT та основні виклики, пов'язані з його безпекою.

Особливу увагу буде приділено огляду криптографічних методів, які застосовуються для захисту протоколів IoT, зокрема принципам використання алгоритму AES в умовах обмежених ресурсів IoT-пристроїв. Наприкінці розділу буде проведено аналіз існуючих методів захисту CoAP, на основі чого буде сформульовано висновки та поставлено задачі для подальшого дослідження.

1.1 Важливість та особливості безпеки IoT пристроїв у сучасних мережах

Інтернет речей (Internet of Things, IoT) стрімко розвивається і впроваджується у численні галузі, починаючи від промисловості та охорони здоров'я до розумних будинків та транспорту. Завдяки IoT-пристроям, які взаємодіють з навколишнім середовищем, користувачі отримують широкий спектр нових можливостей, але одночасно зростає й кількість вразливостей і загроз. Більшість IoT-пристроїв мають обмежені ресурси (обчислювальна потужність, пам'ять, пропускна здатність), що ускладнює використання класичних криптографічних та мережевих протоколів безпеки [1].

Зі зростанням кількості підключених IoT-пристроїв питання їхньої безпеки стає дедалі більш важливим. Безпека IoT-пристроїв критично впливає на захист конфіденційної інформації, стабільність операційних процесів, а також на загальну безпеку інфраструктури. Основні аспекти важливості безпеки IoT-пристроїв включають такі ключові фактори:

IoT-пристрої у багатьох сферах, таких як охорона здоров'я, розумні будинки, споживчі товари, збирають і зберігають чутливі дані користувачів. Це можуть бути особисті дані (ім'я, адреса), медичні показники (частота серцевих скорочень, рівень цукру в крові), інформація про активність і місцезнаходження. У разі компрометації IoT-пристроїв або мереж, зловмисники можуть отримати доступ до цієї конфіденційної інформації, що призводить до ризику витоку даних, шахрайства або крадіжки особистих даних.

Багато IoT-пристроїв є частиною критично важливих інфраструктур, таких як системи охорони здоров'я, транспортні мережі, промислові об'єкти та розумні міста. У цих випадках безперебійна робота пристроїв і захист від кіберзагроз стають питаннями першорядної важливості. Збої в роботі або злом IoT-пристроїв можуть призвести до:

– зупинки операційних процесів;

– відмови в наданні медичних або життєво важливих послуг;

– збитків для підприємств і економіки;

– фізичних пошкоджень інфраструктури або навіть загрози життю людей [1].

IoT-пристрої з обмеженими можливостями часто не мають комплексних систем захисту, що робить їх привабливими цілями для хакерів. Через ці пристрої зловмисники можуть отримати доступ до ширшої мережі, здійснити кібератаки на інші пристрої або мережі, поширювати шкідливе програмне забезпечення або навіть зловживати ресурсами пристрою для атак на інші системи (наприклад, атаки DDoS).

Прикладом є атака Mirai в 2016 році, коли зловмисники скомпрометували сотні тисяч IoT-пристроїв і використали їх для здійснення масової DDoS-атаки на великі інтернет-ресурси, спричинивши значні збої у їхній роботі.

Багато країн уже мають або розробляють нормативні акти та стандарти для забезпечення безпеки IoT-пристроїв. Законодавчі органи все більше зобов'язують виробників та користувачів IoT-пристроїв дотримуватися стандартів захисту даних і кібербезпеки. Це включає вимоги щодо захисту особистих даних користувачів,

вимоги до безпеки промислових мереж та рекомендації щодо управління ризиками в IoT.

Компанії, що не відповідають цим вимогам, можуть бути піддані штрафам, юридичним переслідуванням або навіть забороні на розповсюдження продуктів у певних регіонах.

Різні інциденти з витоками даних та атаками на IoT-пристрої можуть суттєво знизити рівень довіри споживачів до продуктів і брендів, які не дбають про безпеку. Споживачі очікують, що пристрої, які вони використовують, будуть не лише зручними, але й безпечними. Наприклад, якщо у розумних домашніх пристроях, таких як камери безпеки або замки, виявлено вразливості, користувачі можуть повністю відмовитися від таких продуктів або перестати довіряти виробнику. Це може призвести до великих втрат для компанії як з фінансового боку, так і з точки зору репутації [2].

Захист IoT-пристроїв має вирішальне значення для підтримки стабільної та керованої мережі. Більшість IoT-пристроїв є частиною великих мереж, де будь-яка уразливість або компрометація одного пристрою може мати серйозні наслідки для всієї мережі. Уразливості та загрози в мережах IoT можуть призводити до серйозних збоїв, а також ускладнювати моніторинг і управління мережевими активами.

IoT-пристрої часто взаємодіють з іншими пристроями або навіть з іншими типами мереж, такими як Інтернет, мобільні мережі або корпоративні мережі. Це збільшує можливість передачі вразливостей між системами та робить IoT-пристрої привабливими для атак, метою яких є компрометація інших мережевих середовищ.

Безпека IoT-пристроїв є надзвичайно важливою у сучасних умовах, оскільки вони інтегруються у критично важливі системи, збирають конфіденційну інформацію та можуть бути точкою доступу для зловмисників. Інвестування в ефективні засоби безпеки для IoT допомагає захистити конфіденційність даних, забезпечити надійність роботи мереж і захистити репутацію організацій.

Інтернет речей (IoT) має унікальні особливості, які значно впливають на архітектуру безпеки та підходи до захисту. Ці особливості пов'язані з природою

IoT-пристроїв, їхньою взаємодією в мережах та специфічними вимогами до обробки даних. Розглянемо основні фактори, що впливають на безпеку IoT.

IoT-пристрої часто мають обмежену обчислювальну потужність, пам’ять, пропускну здатність та обсяг енергоспоживання, що ускладнює застосування традиційних алгоритмів захисту [2]:

– обмежена обчислювальна потужність: На відміну від ПК чи серверів, IoT-пристрої не можуть виконувати складні криптографічні обчислення без значного впливу на продуктивність. Це знижує ефективність класичних методів шифрування та аутентифікації;

– малий обсяг пам'яті та енергії: Багато IoT-пристроїв мають обмежений обсяг оперативної пам’яті та сховища, а також працюють від батарей. Це обмежує використання методів захисту, які вимагають тривалого зберігання ключів чи регулярного оновлення даних;

– пропускна здатність мережі: Багато IoT-пристроїв працюють у мережах із низькою пропускною здатністю, що обмежує можливості щодо обміну зашифрованими даними або складними запитами аутентифікації.

IoT-пристрої використовують численні протоколи зв'язку для передачі даних, і кожен з них має різні вимоги до захисту:

– відсутність єдиного стандарту: В IoT використовується безліч протоколів, таких як MQTT, CoAP, HTTP, AMQP, Zigbee та Bluetooth. Це ускладнює розробку єдиного рішення з безпеки, яке було б сумісне з усіма протоколами;

– різні рівні безпеки: Кожен протокол має власні методи безпеки, що можуть не забезпечувати повного захисту. Наприклад, протоколи, розроблені для використання у середовищах з низьким енергоспоживанням, можуть мати обмежені функції шифрування чи аутентифікації;

– сумісність і взаємодія: IoT-пристрої часто повинні взаємодіяти з іншими пристроями, що працюють на різних протоколах. Це створює додаткові виклики для інтеграції безпеки, оскільки необхідно забезпечити безпечний обмін даними між протоколами, що не завжди можливо [3].

IoT-пристрої часто розміщуються в місцях, де вони можуть бути доступні для фізичного втручання з боку зловмисників:

– розміщення у відкритих зонах: Багато IoT-пристроїв розташовані у незахищених місцях (наприклад, датчики вуличного освітлення, системи відеоспостереження). Це робить їх вразливими до фізичних атак, наприклад, до роз'єднання чи заміни компонентів;

– можливість фізичного доступу: Зловмисники можуть отримати доступ до пристроїв, підключившись до їхніх портів або інтерфейсів, що дозволяє здійснювати атаки низького рівня, такі як перехоплення даних або навіть зміну прошивки пристрою;

– відсутність захисту від фізичного доступу: Багато IoT-пристроїв не мають фізичного захисту, як-от антивандальні корпуси чи методи захисту від несанкціонованого втручання, що дає можливість зловмисникам впливати на роботу пристроїв або викрасти їх [3].

У IoT-середовищах часто працюють тисячі або навіть мільйони пристроїв, що створює нові виклики для забезпечення безпеки:

– складність управління ключами: Велика кількість підключених пристроїв ускладнює управління криптографічними ключами та ідентифікаційними даними, що створює ризики витоку ключів та компрометації мережі;

– проблеми з централізованим контролем: Через масштабність мереж IoT важко централізовано контролювати безпеку, адже кожен пристрій може вимагати окремого підходу до моніторингу та оновлення безпеки;

– атаки на великі мережі: Вразливість одного IoT-пристрою може призвести до компрометації всієї мережі. Такі атаки, як ботнети (наприклад, Mirai), використовують вразливі IoT-пристрої для розширення масштабів атак і підвищення їхньої ефективності.

Багато IoT-пристроїв працюють у складі розподілених систем, що передбачає обмін даними з іншими пристроями та мережами. Непередбачувані або ненадійні взаємодії між пристроями можуть створювати додаткові вразливості:

– неправильна конфігурація взаємодій: Неправильно налаштовані або конфліктуючі правила взаємодії між пристроями можуть створювати точки входу для атак або спричиняти витоки даних;

– вразливості міжмережевих протоколів: Використання різних протоколів для взаємодії між пристроями (наприклад, між IoT-пристроями та серверами) може відкривати можливості для атак, зокрема MITM (атаки типу «людина посередині»), які можуть перехоплювати або змінювати дані під час передачі;

– автоматизована взаємодія: IoT-пристрої часто мають функції автоматичного з'єднання з іншими пристроями та інфраструктурою, що може відкривати доступ до мережі без необхідної аутентифікації [4].

Можливість регулярного оновлення програмного забезпечення та прошивки пристроїв є важливою умовою безпеки, проте багато IoT-пристроїв мають обмеження в цьому аспекті:

– відсутність оновлень: Багато IoT-пристроїв мають застаріле або заздалегідь встановлене програмне забезпечення, яке не оновлюється. Це підвищує ризик збереження вразливостей, відомих зловмисникам;

– складнощі з доставкою оновлень: Оскільки IoT-пристрої можуть працювати у віддалених місцях або мати обмежений доступ до мережі, оновлення прошивки та безпекових патчів стає проблематичним;

– нестача інфраструктури для автоматичних оновлень: Деякі IoT-пристрої не підтримують автоматичне оновлення програмного забезпечення, що потребує ручного втручання для оновлення, що є трудомістким процесом і може затримувати впровадження безпекових оновлень [4].

Незважаючи на активний розвиток IoT, існує недостатність загальновизнаних стандартів безпеки, що ускладнює розробку уніфікованих рішень:

– відсутність єдиних рекомендацій: Існує велика кількість підходів до безпеки IoT, однак відсутні єдині глобальні стандарти, які б забезпечили комплексний захист усіх пристроїв у мережі;

– різноманітність вимог в різних галузях: Різні сфери використання IoT мають різні вимоги до безпеки (наприклад, медичні пристрої потребують більшого рівня захисту, ніж пристрої для споживачів), що ускладнює створення єдиних протоколів та стандартів;

– низький рівень забезпечення кібербезпеки виробниками: Через високу конкуренцію на ринку деякі виробники можуть ігнорувати вимоги до безпеки, зменшуючи витрати на розробку безпечних пристроїв. Це призводить до появи великої кількості вразливих IoT-пристроїв на ринку.

Різноманітність характеристик IoT-пристроїв накладає суттєві обмеження на ефективність класичних рішень з кібербезпеки. Тому розробка нових методів захисту для IoT потребує врахування специфіки обмежених ресурсів, різноманітності протоколів, фізичної доступності пристроїв та відсутності єдиних стандартів [5].

Для захисту IoT-пристроїв використовуються наступні принципи безпеки:

– конфіденційність: Необхідно забезпечити захист даних, що передаються, від несанкціонованого доступу або перегляду. Це може бути досягнуто за допомогою криптографії та шифрування;

– цілісність даних: Дані, що передаються між IoT-пристроями, повинні залишатися незмінними. Механізми перевірки цілісності дозволяють виявляти випадки несанкціонованих змін або пошкодження інформації під час передачі;

– аутентифікація: Необхідно підтверджувати особу пристрою або користувача перед початком обміну інформацією. Це дозволяє запобігти підключенню сторонніх або підроблених пристроїв до мережі;

– контроль доступу: Обмеження доступу користувачів та пристроїв до мережі та ресурсів дозволяє знизити ризик несанкціонованого доступу та втручання;

– життєвий цикл безпеки: Програмне забезпечення IoT-пристроїв має постійно оновлюватися та підтримуватися розробниками. Підтримка життєвого циклу включає оновлення систем безпеки та захист від нових загроз.

Безпека IoT-пристроїв є критичною у зв’язку з їх важливістю та особливостями функціонування. Приділення уваги безпеці під час розробки, інтеграції та експлуатації IoT-пристроїв дозволяє значно зменшити ризики витоку даних, втрати контролю над пристроями та загрози загальної безпеки системи. Це забезпечує не лише захист конфіденційності та надійність мережі, але й відповідність вимогам регуляторів та збереження довіри користувачів [5].

1.2 Загальна характеристика протоколу CoAP та його роль у мережах IoT

Протокол CoAP (Constrained Application Protocol) є одним із основних протоколів, розроблених спеціально для середовищ з обмеженими ресурсами, що робить його ідеальним для використання в мережах Інтернету речей (IoT). Його було створено Інженерною радою Інтернету (IETF) у межах робочої групи «Core» для забезпечення ефективного обміну даними між пристроями з обмеженою продуктивністю, які працюють у ресурсно обмежених мережах. CoAP використовується для реалізації моделі запит-відповідь, яка схожа на протокол HTTP, але з оптимізацією для IoT середовища.

Протокол CoAP (Constrained Application Protocol) має ряд унікальних характеристик, які роблять його ефективним для використання в мережах Інтернету речей (IoT), особливо в умовах обмежених ресурсів. Цей протокол був спеціально розроблений для спрощеного обміну даними між пристроями з обмеженою обчислювальною потужністю, пам'яттю та енергоспоживанням. Нижче розглянуто основні характеристики CoAP, які забезпечують його функціонування у таких середовищах [5].

Легкість і оптимізація для середовищ з обмеженими ресурсами:

– мінімалістичний дизайн: CoAP має дуже компактну структуру, яка дозволяє знизити обсяги переданих даних, що критично важливо для пристроїв із обмеженою пропускною здатністю та енергоспоживанням. Його заголовок

складається всього з 4 байтів, на відміну від складних протоколів на кшталт HTTP, які мають громіздкі заголовки;

– протокол транспортного рівня UDP: CoAP працює поверх UDP (User Datagram Protocol), а не TCP. Це дозволяє зменшити затримки та обсяг переданих даних, оскільки UDP не передбачає встановлення з'єднання або складних процесів підтвердження, як це робить TCP. В результаті CoAP передає дані швидше та з меншим навантаженням на мережу;

– стиснене кодування даних: Використання стиснення дозволяє зменшити розмір переданих пакетів, що значно знижує навантаження на пристрої з обмеженими ресурсами. Це зменшує витрати на передачу і забезпечує тривалішу роботу пристроїв, які працюють від батарей.

Модель запит-відповідь з додатковим рівнем підтвердження [6].

Аналогія з HTTP: CoAP має модель взаємодії, схожу на HTTP, де клієнт надсилає запит, а сервер надає відповідь. Це дозволяє легко інтегрувати його з існуючими веб-програмами і забезпечити взаємодію між різними пристроями.

Підтвердження та надійність: CoAP вводить дворівневу систему повідомлень:

– Confirmable (підтверджувані) повідомлення – повідомлення, які потребують підтвердження від отримувача. Якщо підтвердження не надійде протягом певного часу, повідомлення надсилається повторно;

– Non-confirmable (непідтверджувані) повідомлення – повідомлення, які не потребують підтвердження, що знижує обсяг трафіку і підходить для передачі даних, де не обов’язково забезпечувати високу надійність (наприклад, періодичні оновлення стану датчика).

Можливість обробки помилок: У разі втрати підтвердження клієнт має можливість надсилати запит повторно, що робить протокол більш надійним для мереж, де можуть траплятися втрати пакетів [6].

Підтримка кешування та спостереження:

– кешування даних: CoAP має вбудовану підтримку кешування, що дозволяє зберігати попередньо отримані дані для повторного використання. Це особливо корисно для зменшення навантаження на мережу, коли клієнти часто запитують однакові дані. Завдяки кешуванню, повторні запити до одного й того ж ресурсу можуть оброблятися значно швидше, без необхідності кожного разу звертатися до кінцевого пристрою.

– механізм спостереження (Observe): CoAP дозволяє клієнтам підписуватися на оновлення певних ресурсів. Це означає, що клієнт отримуватиме повідомлення від сервера щоразу, коли ресурс оновлюється, без необхідності періодично надсилати запити. Цей механізм є надзвичайно ефективним для додатків, де важливо отримувати актуальні дані в реальному часі (наприклад, для моніторингу стану датчиків).

URI-адресація та методи роботи з ресурсами.

Адресація за допомогою URI (Uniform Resource Identifier): Подібно до HTTP, CoAP використовує URI для ідентифікації ресурсів. Це дозволяє забезпечити гнучку та стандартизовану систему адресації ресурсів, що полегшує обмін даними між пристроями. Кожен ресурс має унікальну адресу, яка може бути доступною для різних операцій [7].

Основні методи запитів (GET, POST, PUT, DELETE): CoAP підтримує чотири основні методи, що використовуються також у HTTP:

– GET – запит на отримання ресурсу;

– POST – створення або оновлення ресурсу;

– PUT – заміна або створення ресурсу з певним URI;

– DELETE – видалення ресурсу з заданим URI.

Ці методи забезпечують гнучкість у роботі з ресурсами і дозволяють здійснювати основні CRUD-операції (створення, читання, оновлення, видалення) у середовищах IoT.

Компактна структура заголовків і ефективне використання ресурсів:

– мінімальний розмір заголовка: Структура заголовка CoAP мінімізована до 4 байтів, що значно знижує обсяг трафіку порівняно з HTTP. Це робить CoAP придатним для роботи в умовах обмеженої пропускної здатності та забезпечує ефективне використання мережевих ресурсів. Заголовок містить лише основну інформацію, необхідну для обробки запиту, що підвищує швидкість передачі;

– оптимізація для роботи з малими пакетами даних: Оскільки IoT-пристрої часто обмежені в ресурсах, CoAP налаштований на мінімізацію розмірів пакетів даних. Це особливо важливо для економії енергії, оскільки зменшений розмір повідомлень дозволяє пристроям швидше обробляти та передавати інформацію, що знижує споживання батареї;

– розширення через опції заголовка: CoAP дозволяє додавати додаткову інформацію до заголовків через спеціальні опції, які не впливають на основну структуру повідомлення. Це забезпечує додаткову гнучкість і дозволяє налаштовувати протокол під конкретні потреби, наприклад, для передачі додаткової метаінформації або налаштування параметрів безпеки [7].

Вбудовані механізми безпеки:

– захист за допомогою DTLS (Datagram Transport Layer Security): Для захисту передачі даних CoAP використовує протокол DTLS, який забезпечує шифрування та аутентифікацію даних. DTLS є UDP-еквівалентом TLS і дозволяє захищати дані, які передаються через ненадійні канали, що особливо важливо для бездротових IoT-мереж;

– підтримка аутентифікації і цілісності: CoAP дозволяє впроваджувати механізми аутентифікації пристроїв і перевірки цілісності даних, що дозволяє уникнути несанкціонованого доступу та захистити дані від несанкціонованих змін. Це підвищує загальний рівень захищеності IoT-мереж;

– налаштування часу життя повідомлень: CoAP підтримує встановлення часу життя (TTL) для повідомлень, що підвищує ефективність використання мережевих ресурсів і знижує ризик затримок. Повідомлення, що не досягли своєї мети

протягом певного часу, автоматично відкидаються, що підвищує безпеку і зменшує навантаження на мережу [8].

Основні характеристики CoAP роблять його ідеальним протоколом для IoT-середовищ із обмеженими ресурсами. Завдяки компактній структурі, підтримці кешування, URI-адресації, використанню UDP та можливості спостереження за ресурсами, CoAP забезпечує ефективну взаємодію між пристроями та дозволяє здійснювати захищений і надійний обмін даними. Його легка структура та оптимізація для IoT дозволяють використовувати CoAP у багатьох застосунках, включаючи розумні будинки, промисловий IoT, медичні пристрої та розумні міста.

Протокол CoAP (Constrained Application Protocol) був створений для забезпечення ефективної взаємодії між IoT-пристроями в мережах з обмеженими ресурсами. Завдяки своїй легкій структурі та оптимізації, CoAP надає функціональні можливості, які дозволяють IoT-пристроям ефективно обмінюватися даними, забезпечуючи мінімальне споживання ресурсів. Розглянемо основні функціональні можливості CoAP у контексті IoT [8].

Підтримка багатовузлових мереж (Multicast):

– багатовузлова передача (multicast): CoAP підтримує багатовузлову передачу даних, що дозволяє надсилати один запит одразу кільком пристроям в мережі. Це особливо корисно у ситуаціях, коли необхідно передати однакові дані групі пристроїв одночасно. Наприклад, у системах розумного будинку або промислових мережах IoT, де кілька пристроїв потребують одночасного оновлення або синхронізації;

– оптимізація ресурсів: Використання multicast зменшує навантаження на мережу, оскільки один пакет передається одночасно кільком пристроям, а не кожному окремо. Це знижує трафік та дозволяє ефективніше використовувати мережеві ресурси, що важливо в умовах обмеженої пропускної здатності IoT-мереж;

– застосування в режимах трансляції: У деяких випадках, коли пристрої потребують спільного доступу до даних, багатовузлова передача забезпечує

ефективний спосіб отримання інформації всіма учасниками одночасно. Наприклад, датчики в розумному місті можуть одночасно отримувати інформацію про екологічний стан або погоду [9].

Стиснене кодування і низьке споживання енергії:

– стиснене кодування заголовків: Заголовок CoAP має мінімалістичну структуру, що дозволяє знизити обсяг даних у кожному повідомленні. Використання стисненого кодування забезпечує швидший обмін даними між пристроями і знижує споживання енергії, що важливо для пристроїв, які працюють від батарей;

– знижене навантаження на канали зв'язку: CoAP використовує UDP (User Datagram Protocol), що не вимагає складних процесів підтвердження, як у TCP. Це зменшує затримки і забезпечує швидку передачу даних з мінімальним використанням енергетичних та мережевих ресурсів. Завдяки цьому IoT-пристрої можуть працювати довше від акумуляторів, що є важливим для пристроїв, розташованих у віддалених місцях;

– розширена автономність IoT-пристроїв: Зниження енерговитрат CoAP робить його особливо привабливим для пристроїв, які працюють в автономному режимі. Завдяки мінімальному споживанню енергії такі пристрої можуть довго функціонувати без потреби в регулярній підзарядці, що є важливим для сенсорів і інших пристроїв, розташованих у важкодоступних або автономних місцях [9].

Прості методи аутентифікації і захисту (DTLS):

– використання DTLS (Datagram Transport Layer Security): CoAP інтегрує захист на основі DTLS, що забезпечує шифрування і аутентифікацію повідомлень. DTLS є UDP-версією TLS і забезпечує безпеку передачі даних на транспортному рівні, роблячи передачу інформації безпечною навіть у відкритих мережах. Це дозволяє знижувати ризик несанкціонованого доступу до переданих даних;

– захист даних у режимі реального часу: Використання DTLS дозволяє захищати дані в режимі реального часу, що важливо для додатків, де конфіденційність є критичною, наприклад, в охороні здоров'я або фінансових

додатках. Це дозволяє пристроям обмінюватися чутливою інформацією без ризику втрати конфіденційності;

– контроль цілісності та автентичності повідомлень: DTLS дозволяє забезпечити цілісність повідомлень, що передаються. Це означає, що будь-яка модифікація даних під час передачі буде виявлена, а повідомлення буде відхилено. Крім того, DTLS забезпечує автентифікацію відправника, що дозволяє гарантувати, що дані надходять від довірених пристроїв [10].

Можливість роботи в умовах нестабільного підключення:

– підтвердження отримання: CoAP має вбудовану підтримку Confirmable і Non-confirmable повідомлень. Confirmable повідомлення вимагають підтвердження отримання від отримувача. Це дозволяє пристроям відправляти дані в умовах ненадійного зв’язку та забезпечує можливість повторної відправки у разі, якщо підтвердження не отримано;

– захист від втрати даних: Завдяки механізму повторного відправлення, CoAP може відновлювати передачу даних у разі короткочасних перерв у з'єднанні. Це забезпечує стабільність роботи в умовах непостійного зв’язку, який часто є характерним для бездротових IoT-мереж, таких як мобільні або супутникові з’єднання;

– простота роботи в обмежених мережах: Завдяки своїй легкій структурі, CoAP добре функціонує у мережах із високими затримками або обмеженою пропускною здатністю, дозволяючи підтримувати стабільний обмін даними навіть у складних умовах підключення.

Кешування та спостереження для зменшення навантаження на мережу:

– кешування відповідей: CoAP підтримує кешування, що дозволяє зберігати відповіді на запити для повторного використання. Це зменшує кількість звернень до пристрою, на якому знаходиться ресурс, і відповідно знижує навантаження на мережу. Кешування є особливо корисним у випадках, коли дані часто запитуються багатьма клієнтами, але змінюються рідко;

– механізм спостереження (Observe): CoAP дозволяє клієнтам підписуватися на ресурс для отримання автоматичних оновлень щоразу, коли він змінюється. Це дає змогу пристроям уникати періодичних запитів для перевірки оновлень і отримувати лише актуальну інформацію в реальному часі. Наприклад, датчик температури може надсилати оновлення тільки тоді, коли температура змінюється, замість постійного опитування клієнтом;

– зниження трафіку і енерговитрат: Завдяки кешуванню та спостереженню знижується обсяг переданих даних, що економить пропускну здатність та енергію пристроїв. Це важливо для IoT-мереж з великою кількістю підключених пристроїв, де оптимізація трафіку допомагає зменшити загальне навантаження на мережу [10].

Підтримка взаємодії з іншими протоколами та системами:

– інтеграція з HTTP через проксі-сервери: CoAP може працювати разом з HTTP за допомогою проксі-серверів, які перетворюють HTTP-запити на CoAP-запити і навпаки. Це дозволяє IoT-пристроям інтегруватися з веб-додатками та отримувати доступ до ресурсів, розміщених в інтернеті. Наприклад, система моніторингу може отримувати дані з IoT-пристроїв через HTTP-запити, навіть якщо ці пристрої працюють на CoAP;

– сумісність з іншими IoT-протоколами: CoAP підтримує роботу з іншими IoT-протоколами, такими як MQTT та AMQP, що дозволяє створювати гібридні мережі з різними стандартами зв’язку. Це забезпечує більш гнучку архітектуру для реалізації IoT-систем, де різні пристрої можуть взаємодіяти, використовуючи протоколи, оптимізовані для їхніх потреб;

– простота масштабування: Завдяки підтримці багатовузлової передачі та можливості кешування, CoAP легко масштабується і підтримує розгортання у великих IoT-мережах з тисячами або навіть мільйонами пристроїв. Це дозволяє інтегрувати пристрої у великі мережеві архітектури та забезпечувати їхню безпечну взаємодію [11].

Функціональні можливості CoAP дозволяють забезпечити ефективний і надійний обмін даними в умовах обмежених ресурсів, що є характерним для IoT-

мереж. Завдяки таким функціям, як багатовузлова передача, підтримка кешування, спостереження, інтеграція з іншими протоколами та механізми безпеки DTLS, CoAP забезпечує простий і ефективний спосіб обміну інформацією між IoT-пристроями. Це робить його одним із найзручніших протоколів для використання в різних IoT-середовищах – від розумних будинків і промислових мереж до медичних додатків та інфраструктури розумних міст.

CoAP відіграє важливу роль у забезпеченні комунікації в IoT-мережах з обмеженими ресурсами та надає можливість ефективної взаємодії між пристроями. Сфери, де CoAP є особливо ефективним:

– розумні будинки та побутова автоматизація:

У розумних будинках CoAP дозволяє ефективно передавати дані між пристроями автоматизації, такими як термостати, розумні замки та системи освітлення. Використання CoAP спрощує обмін даними та знижує затримки у відповідях, що підвищує комфорт і безпеку.

– промисловий Інтернет речей (IIoT):

CoAP активно використовується у промислових мережах для моніторингу стану обладнання, керування виробничими процесами та збору даних з датчиків. Завдяки низьким вимогам до ресурсів, CoAP дозволяє інтегрувати IoT в промислові процеси, зберігаючи стабільність і швидкість обміну інформацією [11].

– охорона здоров'я та моніторинг пацієнтів:

У сфері охорони здоров'я CoAP застосовується для віддаленого моніторингу пацієнтів та збору даних з медичних пристроїв. Висока ефективність передачі та низьке енергоспоживання забезпечують тривалу роботу медичних пристроїв без необхідності частого заряджання.

– розумні міста та транспортні системи:

CoAP використовується для зв'язку з датчиками та пристроями управління транспортними та комунальними системами, наприклад, у розумних світлофорах, системах паркування, сміттєзбиральних контейнерах з сенсорами заповненості. Це дозволяє оптимізувати ресурси міста та підвищити якість життя мешканців.

– енергетичний сектор:

CoAP ефективно застосовується для моніторингу споживання енергії та контролю роботи пристроїв у розумних електромережах. Це дає можливість комунальним підприємствам ефективно керувати енергетичними ресурсами та знижувати витрати [11].

Переваги CoAP:

– ефективність: Використання UDP замість TCP дозволяє зменшити затримки та забезпечити швидшу передачу даних;

– підтримка масштабованості: CoAP ефективний у великих мережах з багатьма пристроями завдяки оптимізованій структурі повідомлень;

– можливість кешування: Кешування зменшує навантаження на мережу, що критично важливо для IoT;

– гнучкість у використанні ресурсів: Завдяки простій структурі CoAP підходить для пристроїв з обмеженими обчислювальними ресурсами та енергією.

Недоліки CoAP:

– обмежена підтримка безпеки: Використання DTLS замість традиційного TLS може бути недостатньо надійним для деяких критичних застосувань;

– чутливість до втрат пакетів: Оскільки CoAP працює на основі UDP, вразливість до втрат пакетів може негативно позначитися на цілісності даних;

– складнощі з налаштуванням: Багатокомпонентна структура налаштувань безпеки і стиснення даних може ускладнювати процес впровадження та підтримки.

CoAP є потужним та ефективним протоколом для IoT-мереж завдяки своїй легкій структурі, використанню UDP та оптимізації для обмежених ресурсів. Його можливості щодо кешування, підтвердження отримання та підтримки багатовузлової передачі роблять його ідеальним вибором для різних IoT-застосувань. Однак для певних середовищ може знадобитися розширення його безпекових функцій та підвищення стійкості до втрат даних [12].

1.3 Огляд криптографічних методів захисту протоколів IoT

Криптографія відіграє ключову роль у забезпеченні безпеки протоколів Інтернету речей (IoT), захищаючи дані від несанкціонованого доступу, модифікації та підробки. Оскільки IoT-пристрої часто мають обмежену обчислювальну потужність, пам’ять та енергоспоживання, для їхньої безпеки потрібні легкі та ефективні криптографічні методи. У цьому розділі розглянемо основні криптографічні методи, які використовуються для захисту протоколів IoT, та їхнє застосування.

Симетричне шифрування є одним із найбільш ефективних методів захисту даних для IoT, оскільки воно менш вимогливе до обчислювальних ресурсів у порівнянні з асиметричним шифруванням. У симетричних алгоритмах для шифрування та розшифрування даних використовується один і той самий ключ, що забезпечує швидку обробку та мінімальне споживання енергії [13].

Advanced Encryption Standard (AES):

AES є одним із найпопулярніших алгоритмів симетричного шифрування для IoT, оскільки він поєднує високу швидкість шифрування з надійною безпекою.

Основні режими AES: AES має декілька режимів роботи, які використовуються в IoT, наприклад:

– ECB (Electronic Codebook) – простий, але вразливий до аналізу шаблонів;

– CBC (Cipher Block Chaining) – забезпечує кращу захищеність, адже кожен блок даних залежить від попереднього;

– CTR (Counter) – дозволяє паралельне шифрування блоків, що підходить для високонавантажених IoT-пристроїв.

Використання в IoT: AES широко застосовується у протоколах IoT, таких як CoAP та MQTT, для шифрування передачі даних. Завдяки своїй ефективності AES підходить для пристроїв з обмеженими ресурсами, таких як сенсори, трекери та розумні побутові прилади [13].

Stream Ciphers:

– потокові шифри шифрують дані потоком, обробляючи кожен біт або байт індивідуально, що дозволяє досягти високої швидкості шифрування при мінімальному використанні ресурсів;

– RC4 та ChaCha20 – популярні потокові шифри, які використовуються для шифрування даних у IoT. ChaCha20, зокрема, має високу ефективність і забезпечує кращу безпеку, ніж RC4;

– застосування в IoT: Потокові шифри є ефективними для сенсорів і мікроконтролерів, які повинні швидко шифрувати дані з мінімальним навантаженням на процесор і пам’ять [14].

Blowfish і її полегшена версія Twofish:

– Blowfish є ще одним популярним симетричним алгоритмом, який забезпечує високий рівень безпеки і працює швидко на IoT-пристроях. Twofish є покращеною версією Blowfish з підвищеною безпекою;

– особливості: Ці алгоритми можуть використовуватися у пристроях IoT, де потрібно захистити дані без значного навантаження на систему.

Симетричне шифрування є одним із найбільш ефективних методів захисту даних для IoT, оскільки воно менш вимогливе до обчислювальних ресурсів у порівнянні з асиметричним шифруванням. У симетричних алгоритмах для шифрування та розшифрування даних використовується один і той самий ключ, що забезпечує швидку обробку та мінімальне споживання енергії.

Advanced Encryption Standard (AES):

AES є одним із найпопулярніших алгоритмів симетричного шифрування для IoT, оскільки він поєднує високу швидкість шифрування з надійною безпекою.

Основні режими AES: AES має декілька режимів роботи, які використовуються в IoT, наприклад:

– ECB (Electronic Codebook) – простий, але вразливий до аналізу шаблонів.

– CBC (Cipher Block Chaining) – забезпечує кращу захищеність, адже кожен блок даних залежить від попереднього.

– CTR (Counter) – дозволяє паралельне шифрування блоків, що підходить для високонавантажених IoT-пристроїв [14].

Використання в IoT: AES широко застосовується у протоколах IoT, таких як CoAP та MQTT, для шифрування передачі даних. Завдяки своїй ефективності AES підходить для пристроїв з обмеженими ресурсами, таких як сенсори, трекери та розумні побутові прилади.

Stream Ciphers:

– потокові шифри шифрують дані потоком, обробляючи кожен біт або байт індивідуально, що дозволяє досягти високої швидкості шифрування при мінімальному використанні ресурсів;

– RC4 та ChaCha20 – популярні потокові шифри, які використовуються для шифрування даних у IoT. ChaCha20, зокрема, має високу ефективність і забезпечує кращу безпеку, ніж RC4;

– застосування в IoT: Потокові шифри є ефективними для сенсорів і мікроконтролерів, які повинні швидко шифрувати дані з мінімальним навантаженням на процесор і пам’ять.

Blowfish і її полегшена версія Twofish:

– Blowfish є ще одним популярним симетричним алгоритмом, який забезпечує високий рівень безпеки і працює швидко на IoT-пристроях. Twofish є покращеною версією Blowfish з підвищеною безпекою.

– особливості: Ці алгоритми можуть використовуватися у пристроях IoT, де потрібно захистити дані без значного навантаження на систему.

Асиметричне шифрування використовує два різні ключі: відкритий ключ для шифрування та закритий ключ для розшифрування. Це забезпечує високий рівень безпеки, але є більш ресурсомістким порівняно з симетричним шифруванням, тому асиметричні методи часто використовуються для аутентифікації та обміну ключами, але рідко для безпосереднього шифрування даних у IoT [15].

– RSA (Rivest-Shamir-Adleman):

RSA – класичний алгоритм асиметричного шифрування, який забезпечує високий рівень захисту і широко використовується для захисту даних і автентифікації. Однак RSA потребує значних обчислювальних ресурсів, тому його застосування в IoT обмежене.

Обмін ключами та сертифікація: RSA може використовуватися для обміну ключами між пристроями або підтвердження автентичності через сертифікати, наприклад, у протоколі DTLS.

– ECC (Elliptic Curve Cryptography):

ECC є більш ефективним алгоритмом порівняно з RSA, оскільки забезпечує високий рівень безпеки з використанням коротших ключів (наприклад, 256-бітний ключ ECC забезпечує безпеку, подібну до 3072-бітного ключа RSA).

Застосування в IoT: ECC використовується для аутентифікації пристроїв, обміну ключами та шифрування. Завдяки своїй ефективності ECC ідеально підходить для пристроїв IoT з обмеженими ресурсами, таких як смарт-годинники, термостати або сенсори [15].

Варіанти ECC: Відомі реалізації ECC включають ECDSA (Elliptic Curve Digital Signature Algorithm) для цифрових підписів і ECDH (Elliptic Curve Diffie-Hellman) для обміну ключами, що широко використовуються в IoT.

– MQV (Menezes-Qu-Vanstone):

MQV є розширеним варіантом алгоритму обміну ключами на основі еліптичних кривих, що забезпечує додатковий рівень захисту, зокрема проти атак на повторення. Він може застосовуватися в IoT для захисту зв'язку між пристроями.

Гібридні методи поєднують симетричне та асиметричне шифрування для забезпечення безпечного обміну даними. Асиметричне шифрування використовується для передачі симетричного ключа, який потім застосовується для шифрування всього каналу.

Протокол SSL/TLS та DTLS (для UDP):

– TLS забезпечує шифрування каналу передачі даних між клієнтом і сервером та використовується для захисту з'єднань через TCP, тоді як DTLS

(Datagram Transport Layer Security) є UDP-аналогом TLS і широко використовується у протоколах IoT, таких як CoAP;

– гібридний підхід: На початку з'єднання за допомогою асиметричного шифрування обмінюються ключами, після чого для шифрування даних використовується симетричний алгоритм. Це забезпечує високу швидкість передачі та безпеку;

– застосування в IoT: DTLS використовується в CoAP для захисту повідомлень, забезпечуючи шифрування, цілісність і аутентифікацію даних, що передаються [15].

Протокол IPsec (Internet Protocol Security):

– IPsec забезпечує захищеність на рівні мережі та використовує гібридне шифрування для захисту IP-пакетів. Він створює захищені тунелі між пристроями, забезпечуючи конфіденційність і цілісність даних;

– застосування в IoT: IPsec може використовуватися для захисту IoT-пристроїв у мережах із більшою пропускною здатністю, наприклад, для шлюзів та хмарних серверів, які керують великими обсягами IoT-даних.

Для забезпечення автентичності та цілісності даних у IoT-пристроях використовуються цифрові підписи та хешування. Ці методи дозволяють перевірити, що дані надійшли від довіреного джерела і не були змінені під час передачі.

Хеш-функції (MD5, SHA-2, SHA-3):

– MD5 і SHA-2 використовуються для перевірки цілісності даних, але MD5 вважається застарілим, оскільки у ньому були виявлені вразливості;

– SHA-2 і SHA-3 забезпечують більш надійну цілісність даних і широко застосовуються у протоколах IoT для виявлення змін у переданих даних;

– застосування в IoT: Хеш-функції використовуються для виявлення маніпуляцій з даними і перевірки їхньої цілісності в промислових IoT-системах, медичних приладах і сенсорних мережах [16].

Цифровий підпис на основі ECC (ECDSA):

– ECDSA (Elliptic Curve Digital Signature Algorithm) використовує еліптичні криві для створення цифрових підписів. Цей алгоритм дозволяє підтвердити автентичність даних, забезпечуючи високий рівень захисту навіть при використанні коротких ключів;

– застосування в IoT: ECDSA дозволяє підтверджувати автентичність даних від різних IoT-пристроїв, таких як сенсори або промислові контролери, забезпечуючи їхню безпеку при мінімальному навантаженні на пристрій.

Захист IoT-пристроїв потребує ефективної системи аутентифікації та управління криптографічними ключами. Розглянемо основні методи, що використовуються в IoT:

Протокол Kerberos:

– Kerberos – це мережевий протокол аутентифікації, що забезпечує надійну ідентифікацію учасників обміну інформацією. У IoT він може використовуватися для безпечного обміну ключами.

PKI (Public Key Infrastructure):

– PKI забезпечує управління сертифікатами та ключами для аутентифікації пристроїв. У IoT PKI дозволяє пристроям отримувати сертифікати для встановлення безпечних з’єднань із сервером або між собою.

Менеджмент ключів на базі ECC:

– використання ECC для управління ключами дозволяє створювати компактні та ефективні сховища ключів, зменшуючи ресурсні витрати, що є критичним для пристроїв IoT з обмеженою пам’яттю [16].

Криптографічні методи відіграють вирішальну роль у забезпеченні безпеки IoT-пристроїв. Симетричні та асиметричні методи шифрування, гібридні протоколи, а також цифрові підписи та аутентифікація дозволяють ефективно захищати дані, передані між IoT-пристроями. Специфіка IoT-середовища, така як обмеженість ресурсів, вимагає оптимізованих криптографічних рішень, зокрема AES, ECC та DTLS, які забезпечують конфіденційність, цілісність і доступність даних навіть у середовищах з обмеженими ресурсами.

1.4 Принципи використання алгоритму AES в умовах IoT

Алгоритм AES (Advanced Encryption Standard) є одним з найпопулярніших та надійних симетричних алгоритмів шифрування і широко використовується в умовах Інтернету речей (IoT) завдяки своїй ефективності та високій продуктивності. Оскільки AES забезпечує високий рівень захисту при відносно низькому споживанні ресурсів, він підходить для IoT-пристроїв, які зазвичай мають обмежену обчислювальну потужність і пам’ять. У цьому розділі розглянемо ключові принципи використання алгоритму AES в умовах IoT, включаючи вибір режиму роботи, управління ключами та оптимізацію продуктивності.

AES – це симетричний блоковий алгоритм шифрування, де один і той самий ключ використовується для шифрування та розшифрування даних. Він підтримує різні розміри ключів (128, 192 та 256 бітів) і працює з блоками даних розміром 128 бітів. Алгоритм AES включає кілька раундів, у яких здійснюються різні математичні операції над даними, забезпечуючи надійне шифрування [17].

– безпека AES: Завдяки стійкості до різних типів криптоаналітичних атак AES вважається одним із найбезпечніших алгоритмів. Використання 128-бітного або 256-бітного ключа забезпечує високу криптографічну стійкість, що особливо важливо для захисту IoT-пристроїв;

– ефективність роботи: AES використовує низку простих операцій, таких як заміна, перестановка та побітові операції, які можуть виконуватися швидко і ефективно навіть на пристроях з обмеженими ресурсами.

Щоб забезпечити максимальну продуктивність і мінімальне споживання енергії, алгоритм AES в IoT-середовищах часто оптимізується.

Аппаратна реалізація:

– AES може бути реалізований у вигляді апаратних модулів (AES-акселератори), які забезпечують значно вищу швидкість шифрування при меншому навантаженні на основний процесор;

– багато сучасних мікроконтролерів для IoT мають вбудовану підтримку AES, що дозволяє виконувати шифрування на апаратному рівні і значно знижує споживання енергії.

Зниження кількості раундів шифрування для специфічних застосувань:

– у деяких IoT-додатках, де надмірна безпека не є критичною, кількість раундів AES може бути зменшена для зменшення навантаження. Наприклад, замість стандартних 10 раундів для 128-бітного ключа, можна використовувати менше, якщо безпека має бути збалансована з продуктивністю;

– цей підхід може бути застосований у низькорівневих пристроях, де надмірна безпека не є пріоритетом [17].

Використання потокового шифрування на основі AES (AES-CTR):

– потоковий режим шифрування, як-от AES-CTR, дозволяє паралельне обчислення блоків і знижує затримки. Це підвищує ефективність обміну даними в реальному часі, що особливо важливо для IoT-пристроїв з обмеженими ресурсами;

– потокове шифрування може забезпечити постійний потік даних з низькою затримкою, що підходить для додатків реального часу, таких як датчики та моніторингові системи.

Мінімізація передачі даних:

– одним зі способів підвищення продуктивності є шифрування лише критично важливих даних, замість усього обсягу інформації. Це дозволяє зменшити обсяг даних, що шифруються, і, відповідно, оптимізувати споживання ресурсів;

– наприклад, у медичних пристроях або розумних сенсорах можна шифрувати тільки чутливі параметри, такі як ідентифікація пацієнта або конфіденційні медичні дані [18].

Приклади застосування AES в IoT

– захист конфіденційності даних у розумних будинках:

AES забезпечує конфіденційність у розумних пристроях, таких як камери відеоспостереження, розумні замки або термостати, дозволяючи захистити дані від перехоплення зловмисниками.

– безпечна передача медичних даних:

У пристроях для моніторингу здоров'я, таких як трекери, датчики життєвих показників або медичні імпланти, AES забезпечує захист чутливих даних пацієнтів під час передачі на сервер або до медичних установ.

– захист даних в промислових системах IIoT:

У промислових IoT-системах, де збираються та аналізуються дані з різних сенсорів і пристроїв, AES забезпечує конфіденційність інформації, наприклад, про параметри роботи обладнання, температурні показники, стан вібрації тощо [18].

Алгоритм AES є основним методом шифрування для IoT завдяки своїй високій ефективності, безпеці та гнучкості в налаштуваннях. Правильний вибір режиму роботи AES, належне управління ключами та оптимізація продуктивності дозволяють забезпечити надійний захист даних при мінімальному навантаженні на ресурси IoT-пристроїв. Це робить AES незамінним у таких галузях, як розумні будинки, охорона здоров'я та промислові IoT-системи, де конфіденційність, цілісність та швидкість обробки є критичними параметрами для забезпечення безпечної передачі даних.

1.5 Аналіз існуючих методів захисту CoAP у мережах IoT

Протокол CoAP (Constrained Application Protocol) був розроблений для комунікації між IoT-пристроями, що мають обмежені ресурси. Для забезпечення конфіденційності, цілісності та автентичності даних, що передаються через CoAP, існує низка методів захисту. У цьому розділі розглянемо та проаналізуємо існуючі методи захисту CoAP, їхні переваги, недоліки та сферу застосування в мережах IoT.

DTLS (Datagram Transport Layer Security) є основним методом захисту для CoAP, оскільки забезпечує шифрування і автентифікацію на транспортному рівні, аналогічно TLS для TCP. DTLS працює поверх UDP, на якому базується CoAP,

забезпечуючи захист без створення надмірного навантаження на ресурси пристрою [19].

Механізми безпеки DTLS:

– шифрування даних: DTLS використовує симетричні алгоритми шифрування, наприклад, AES, для забезпечення конфіденційності переданих даних;

– цілісність даних: DTLS застосовує HMAC (Hash-based Message Authentication Code) для перевірки цілісності даних, що дозволяє виявляти будь-які зміни, внесені в повідомлення під час передачі;

– аутентифікація пристроїв: DTLS підтримує автентифікацію за допомогою сертифікатів або попередньо спільних ключів, що дозволяє переконатися в тому, що дані надходять від довіреного пристрою.

Переваги DTLS:

– стійкий захист: DTLS забезпечує високий рівень захисту, який відповідає вимогам безпеки для IoT-мереж, включаючи шифрування, автентифікацію та перевірку цілісності;

– сумісність з CoAP: Оскільки DTLS працює поверх UDP, він може бути легко інтегрований з CoAP, не порушуючи його роботи.

Недоліки DTLS:

– затримки і споживання ресурсів: DTLS додає додаткову затримку і навантаження на обчислювальні ресурси під час встановлення з’єднання, що може бути критичним для IoT-пристроїв з обмеженими можливостями;

– проблеми з рукопотисканням: Процес встановлення з’єднання DTLS (рукопотискання) вимагає кількох обмінів повідомленнями, що може викликати проблеми у нестабільних або низькопродуктивних мережах [19].

Застосування в IoT:

– DTLS підходить для застосувань, де потрібен високий рівень безпеки, наприклад, у медичних або промислових IoT-системах, де забезпечення конфіденційності та цілісності даних є критично важливим.

Попередньо спільний ключ (PSK) є методом автентифікації, при якому CoAP-пристрої використовують заздалегідь встановлений ключ для автентифікації один одного. PSK забезпечує високий рівень захисту з мінімальним навантаженням на ресурси, що робить його привабливим для пристроїв IoT з обмеженими можливостями.

Принципи роботи PSK:

– кожен пристрій має унікальний ключ, відомий обом сторонам, який використовується для аутентифікації під час передачі даних. Шифрування та автентифікація повідомлень забезпечуються з використанням цього ключа;

– PSK може застосовуватися як у поєднанні з DTLS, так і у спрощених протоколах для захисту сеансів CoAP [20].

Переваги PSK:

– низьке навантаження на ресурси: PSK потребує мінімум ресурсів для обробки, оскільки не потребує складних криптографічних обчислень;

– швидкість встановлення з’єднання: PSK дозволяє встановлювати з’єднання швидше порівняно з іншими методами, такими як сертифікати.

Недоліки PSK:

– проблеми з управлінням ключами: У великих мережах IoT може бути складно керувати великою кількістю унікальних ключів для кожного пристрою;

– компрометація одного ключа: Якщо один PSK-ключ буде скомпрометований, це створить серйозний ризик для всієї мережі, оскільки зловмисник зможе отримати доступ до захищених даних.

Застосування в IoT:

– PSK є популярним вибором для невеликих мереж з обмеженою кількістю пристроїв, а також для систем, де ресурси є критичним обмеженням, наприклад, у розумних сенсорних мережах або контролерах.

OSCORE (Object Security for Constrained RESTful Environments) – це метод захисту на рівні прикладного шару, який шифрує самі повідомлення CoAP [21].

OSCORE забезпечує захист, незалежний від транспортного рівня, що робить його стійким до атак на нижніх рівнях протоколу.

Механізми безпеки OSCORE:

– енд-то-енд шифрування: OSCORE дозволяє шифрувати дані на рівні додатку, забезпечуючи захист повідомлень навіть при транзитному з’єднанні через проксі-сервери;

– цілісність і автентичність: OSCORE забезпечує перевірку цілісності повідомлень і автентифікацію відправника, що запобігає спробам модифікації даних.

Переваги OSCORE:

– стійкість до атак: OSCORE захищає дані на рівні об’єкта, що знижує ймовірність атак на рівні мережі;

– незалежність від транспортного рівня: Оскільки OSCORE захищає дані на рівні прикладного шару, він може працювати з будь-якими транспортними протоколами, що особливо корисно для мультипротокольних середовищ.

Недоліки OSCORE:

– складність реалізації: Реалізація OSCORE вимагає більш складних обчислювальних ресурсів порівняно з PSK, що може створювати додаткове навантаження на IoT-пристрої з дуже обмеженими ресурсами;

– інтеграція з існуючими системами: Використання OSCORE вимагає змін у додатках, оскільки він працює на рівні повідомлень CoAP, що може бути складно в існуючих системах [22].

Застосування в IoT:

– OSCORE підходить для великих IoT-систем, які потребують стійкого захисту незалежно від транспортного середовища, таких як промислові мережі та розподілені системи моніторингу.

Для забезпечення цілісності та автентичності даних у CoAP можуть використовуватися методи хешування та цифрові підписи.

Алгоритми хешування (SHA-256, SHA-3):

– SHA-256 і SHA-3 широко застосовуються для створення хешів повідомлень CoAP, що дозволяє перевіряти цілісність даних;

– застосування хеш-функцій: Хеш-функції використовуються для перевірки цілісності даних, а також для створення HMAC, який поєднує хешування з секретним ключем для забезпечення автентичності.

Цифрові підписи на основі ECC (Elliptic Curve Cryptography):

– використання ECDSA (Elliptic Curve Digital Signature Algorithm) дозволяє створювати цифрові підписи для даних CoAP, що дозволяє отримувачеві переконатися у їхній автентичності;

– застосування в IoT: Цифрові підписи підходять для IoT-пристроїв, що передають критичні дані, наприклад, у медичних або промислових системах, де важлива гарантія автентичності [22].

Переваги хешування і підписів:

– гарантія цілісності: Хешування дозволяє виявити спроби зміни даних;

– автентифікація відправника: Цифрові підписи дозволяють підтвердити, що повідомлення CoAP надійшло від довіреного відправника.

Недоліки хешування і підписів:

– ресурсомісткість: Процеси хешування та створення підписів вимагають значних обчислювальних ресурсів, що може створити навантаження на слабких IoT-пристроях;

– затримки при передачі: Підписи можуть збільшувати затримки під час обробки повідомлень.

PKI (Public Key Infrastructure) використовується для управління сертифікатами та криптографічними ключами в мережах IoT, що дозволяє забезпечити аутентифікацію пристроїв [23].

Принципи PKI:

– PKI використовує інфраструктуру публічних ключів для випуску, зберігання та перевірки сертифікатів, що дозволяє кожному пристрою отримати унікальний сертифікат, який підтверджує його ідентичність;

– сертифікати надаються центральним сертифікаційним центром (CA), що підвищує рівень безпеки в мережі.

Переваги PKI:

– надійна автентифікація: Сертифікати забезпечують високий рівень довіри і гарантують, що тільки авторизовані пристрої можуть підключатися до мережі;

– масштабованість: PKI добре підходить для великих IoT-мереж, де необхідно аутентифікувати велику кількість пристроїв.

Недоліки PKI:

– складність управління сертифікатами: PKI вимагає ефективного управління сертифікатами, що може бути проблематично для великих мереж IoT;

– витрати на обчислення: Верифікація сертифікатів вимагає ресурсів, що може перевищувати можливості деяких IoT-пристроїв.

Застосування в IoT:

– PKI підходить для захищених промислових і комерційних мереж, де необхідно забезпечити високу надійність та захищеність комунікацій між пристроями.

Для захисту протоколу CoAP у мережах IoT використовуються різні методи, кожен з яких має свої унікальні особливості, переваги та обмеження. Нижче представлено порівняльний аналіз методів захисту, що дозволяє оцінити їхню ефективність та доцільність використання залежно від конкретних вимог IoT-мережі [24].

У таблиці 1.1 показано, як різні методи забезпечують ключові характеристики безпеки, наскільки вони ресурсомісткі, для яких типів IoT-мереж підходять та які мають обмеження. Це дозволяє підібрати оптимальне рішення для захисту CoAP, враховуючи специфічні вимоги до безпеки, обмеження ресурсів і масштаби мережі.

Таблиця 1.1 – Порівняння методів захисту IoT-мереж

Метод захисту	Характеристики безпеки	Ресурсоспоживання	Застосування	Недоліки
DTLS (Datagram Transport Layer Security)	Шифрування, автентифікація, цілісність даних	Високе, особливо під час встановлення з'єднання	Критичні IoT-системи (медичні, промислові)	Додаткові затримки та навантаження на ресурси
PSK (Pre-Shared Key)	Автентифікація та шифрування через попередньо встановлений ключ	Низьке	Невеликі та обмежені мережі IoT (сенсорні)	Труднощі управління ключами в масштабованих мережах
OSCORE (Object Security for Constrained RESTful Environments)	Енд-то-енд шифрування та автентифікація на рівні додатка	Середнє, залежить від реалізації	Великі IoT-системи, промислові та захищені сенсорні мережі	Вимоги до ресурсів, складність інтеграції
Хешування та цифровий підпис	Цілісність і автентифікація даних	Середнє-високе, залежить від алгоритму	Критичні IoT-системи (медичні, промислові)	Ресурсомісткість, можливі затримки
PKI (Public Key Infrastructure)	Сертифікатна автентифікація	Високе, особливо для перевірки сертифікатів	Великі мережі IoT (розподілені системи, промислові мережі)	Складне управління сертифікатами

DTLS забезпечує високий рівень захисту, але може перевантажувати пристрої з обмеженими ресурсами через затрати під час встановлення з’єднання.

PSK має низьке ресурсоспоживання і швидко встановлює з’єднання, але важко масштабувати у великих мережах.

OSCORE забезпечує енд-то-енд захист незалежно від транспортного рівня, але потребує більше ресурсів та складний в інтеграції.

Хешування та цифровий підпис добре підходять для цілісності і автентифікації, але можуть бути ресурсомісткими для слабких пристроїв.

PKI є найнадійнішим для масштабованих мереж, але його складне управління сертифікатами може бути проблематичним для IoT [25].

Це порівняння допомагає оцінити методи захисту CoAP з точки зору придатності до застосування в різних типах IoT-мереж.

Існуючі методи захисту CoAP забезпечують широкий спектр механізмів безпеки для різних типів IoT-застосувань, дозволяючи пристроям функціонувати в умовах обмежених ресурсів. DTLS надає комплексне рішення для захисту на транспортному рівні, OSCORE – на рівні додатка, а PSK і PKI використовуються для автентифікації залежно від вимог до ресурсів і масштабу мережі. Хешування і цифрові підписи дозволяють забезпечити цілісність і автентичність даних. Вибір конкретного методу захисту CoAP залежить від вимог до продуктивності, ресурсів пристроїв та критичності даних у конкретному IoT-додатку.

1.6 Висновки та постановка задачі

У цій роботі проведено дослідження теоретичних засад захисту протоколу CoAP для IoT-пристроїв. Було детально проаналізовано різні аспекти безпеки протоколу CoAP, що включають важливість захисту даних у IoT-мережах, основні криптографічні методи, принципи використання алгоритму AES, а також існуючі методи захисту CoAP, їхні переваги та недоліки. Дослідження дозволило отримати розгорнуте розуміння ключових вимог і підходів до захисту протоколів у ресурсно обмежених IoT-середовищах, що є критичним для забезпечення конфіденційності, цілісності та доступності даних у таких мережах.

На основі проведеного дослідження та аналізу сформульовано такі завдання для подальшої розробки:

– розробка алгоритму шифрування протоколу CoAP на основі AES із динамічно змінюваними ключами;

– проектування механізмів аутентифікації повідомлень для CoAP у IoT-мережах;

– розробка схеми управління життєвим циклом ключів для забезпечення додаткового рівня безпеки;

– практична реалізація та тестування запропонованих рішень для захисту протоколу CoAP у реальному IoT-середовищі.

Ці завдання спрямовані на вдосконалення безпеки протоколу CoAP та забезпечення захисту IoT-пристроїв від потенційних загроз у сучасних мережах.

2 СТВОРЕННЯ ВДОСКОНАЛЕНОГО ЗАХИСТУ ПРОТОКОЛУ COAP ДЛЯ IOT ПРИСТРОЇВ ЗА ДОПОМОГОЮ КРИПТОГРАФІЧНОГО АЛГОРИТМУ AES З ДИНАМІЧНО ЗМІНЮВАНИМИ КЛЮЧАМИ НА ОСНОВІ ХАРАКТЕРИСТИК МЕРЕЖІ, МЕХАНІЗМІВ АУТЕНТИФІКАЦІЇ ПОВІДОМЛЕНЬ ТА ОБМЕЖЕННЯ ЧАСУ ЖИТТЯ КЛЮЧІВ

У цьому розділі буде представлено процес створення вдосконаленого захисту протоколу CoAP для IoT-пристроїв із використанням криптографічного алгоритму AES. Основна увага приділятиметься реалізації динамічно змінюваних ключів на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів.

Спочатку буде розглянуто особливості впровадження вдосконаленого захисту протоколу CoAP, враховуючи обмежені ресурси IoT-пристроїв та специфіку їхньої взаємодії в мережах. Далі буде детально проаналізовано підхід до забезпечення безпеки за допомогою алгоритму AES із динамічним управлінням ключами, а також описано механізми аутентифікації та перевірки цілісності повідомлень.

У межах розділу також буде розроблено алгоритм взаємодії між IoT-пристроями за допомогою вдосконаленого протоколу, що забезпечує захищений обмін даними. Завершуючи розділ, будуть сформульовані висновки щодо реалізації запропонованого підходу та його переваг у порівнянні з існуючими методами.

2.1 Особливості створення вдосконаленого захисту протоколу coap для IoT пристроїв за допомогою криптографічного

Протокол CoAP (Constrained Application Protocol) широко використовується в IoT-середовищах, де пристрої з обмеженими ресурсами обмінюються інформацією. Для забезпечення захисту таких мереж потрібні оптимізовані криптографічні підходи, що враховують обмеження обчислювальної потужності, пам’яті, енергоспоживання та потребу в безперервному з'єднанні. Вдосконалення захисту CoAP передбачає застосування легких і ефективних методів шифрування,

аутентифікації та управління ключами. У цьому підрозділі буде розглянуто основні особливості, вимоги та виклики створення вдосконаленого захисту CoAP за допомогою криптографічних методів [26].

Основні вимоги до захисту CoAP у IoT-середовищах включають кілька критичних аспектів, які забезпечують надійний і безпечний обмін даними між пристроями з обмеженими ресурсами. Однією з ключових вимог є забезпечення конфіденційності інформації, яка передається між пристроями. У IoT-мережах часто передаються чутливі або персональні дані, такі як показники здоров'я чи дані про місцезнаходження користувача, що потребує захисту від несанкціонованого доступу. Для цього використовується шифрування даних, яке гарантує, що навіть у разі перехоплення дані залишаться недоступними для сторонніх осіб.

Ще однією важливою вимогою є забезпечення цілісності даних, що дозволяє захищати інформацію від будь-яких спроб несанкціонованої зміни або підробки під час передачі. Збереження цілісності гарантує, що повідомлення, відправлене одним пристроєм, досягне іншого у тому ж вигляді, в якому воно було відправлене, без будь-яких змін. Це особливо важливо для IoT-середовищ, де спотворення даних може призвести до помилкових рішень або збоїв у роботі мережі.

Автентифікація відправника та отримувача є ще однією необхідною вимогою для захисту CoAP, оскільки вона дозволяє переконатися, що обидві сторони обміну даними є довіреними і дійсними. Завдяки автентифікації можна запобігти випадкам несанкціонованого доступу та гарантувати, що повідомлення надходять лише від авторизованих джерел. У середовищах, де безпека є критичною, ця вимога стає основним захисним елементом.

Додатково CoAP у IoT-середовищах потребує захисту від повторних атак, які полягають у тому, що зловмисник перехоплює повідомлення і надсилає його повторно, створюючи ілюзію справжньої комунікації. Для захисту від цього типу атак CoAP потребує динамічного управління ключами або контролю часу дійсності сесійних ключів, що запобігає використанню одних і тих самих даних для кількох з'єднань [27].

Нарешті, усі методи захисту в CoAP мають бути оптимізовані для мінімального споживання ресурсів, адже IoT-пристрої зазвичай обмежені в обчислювальній потужності, пам’яті та енергетичних можливостях. Це означає, що методи шифрування, автентифікації та перевірки цілісності даних повинні бути не лише ефективними, але й легкими, щоб не впливати на тривалість роботи пристроїв. Важливість таких вимог у IoT-середовищах пояснюється необхідністю забезпечення надійного захисту за умов, коли обчислювальні ресурси обмежені, а підключення може бути нестабільним [28].

Механізми аутентифікації та управління ключами є основними компонентами захисту протоколу CoAP у IoT-середовищах, забезпечуючи надійний і захищений обмін даними між пристроями. У середовищах з обмеженими ресурсами потрібно використовувати такі методи аутентифікації, які не лише гарантують безпеку, але й мінімізують навантаження на пристрої. Одним із найпоширеніших і ефективних методів є використання попередньо спільного ключа (PSK). PSK підходить для невеликих мереж IoT, де всі пристрої можуть використовувати один заздалегідь встановлений ключ для взаємної автентифікації. Цей метод є простим і потребує мінімальних обчислювальних ресурсів, що дозволяє знизити навантаження на пристрої з обмеженими можливостями. Однак PSK не підходить для масштабованих систем, оскільки використання одного ключа в усій мережі може підвищити ризик компрометації — компрометація одного ключа поставить під загрозу всю мережу [28].

Іншим механізмом аутентифікації, який є більш надійним у масштабованих IoT-мережах, є аутентифікація на основі сертифікатів, що реалізується з використанням інфраструктури відкритих ключів (PKI). PKI дозволяє кожному пристрою мати унікальний цифровий сертифікат, що підтверджує його ідентичність. Такий сертифікат видається централізованим сертифікаційним центром (CA), що забезпечує високий рівень довіри і дозволяє гарантувати, що лише авторизовані пристрої можуть підключатися до мережі. PKI є особливо корисним для великих розподілених IoT-мереж, де потрібно забезпечити надійну автентифікацію великої кількості пристроїв. Хоча PKI надає високу безпеку, його

реалізація потребує більше ресурсів для управління сертифікатами, а також для виконання криптографічних операцій, що може перевищувати можливості малопотужних IoT-пристроїв [29].

Крім аутентифікації, в IoT важливу роль відіграє управління життєвим циклом ключів, яке передбачає регулярне оновлення ключів для забезпечення захисту протоколу CoAP. Використання одного і того ж ключа протягом тривалого часу підвищує ризик його компрометації. Тому життєвий цикл ключа може бути обмеженим як за часом, так і за кількістю використань. Це дозволяє періодично замінювати старий ключ на новий, знижуючи ймовірність успішної атаки на ключ. Оновлення ключів може здійснюватися автоматично або ініціюватися за умовами, наприклад, після встановленої кількості сесій або за заданим інтервалом часу.

Одним з ефективних способів автоматичного оновлення ключів є використання алгоритму Диффі-Геллмана (Diffie-Hellman, DH) або його варіантів, які дозволяють безпечно обмінюватися ключами навіть через незахищені канали. DH дозволяє кожному пристрою генерувати свій унікальний ключ для сеансу зв'язку, що потім узгоджується з іншою стороною без необхідності передачі самого ключа. Цей механізм особливо корисний для мереж IoT, де необхідно підтримувати високий рівень безпеки без значного навантаження на канал зв'язку [29].

Таким чином, різні механізми аутентифікації та управління ключами забезпечують гнучкість і надійність захисту CoAP у IoT-середовищах. Використання PSK підходить для малих, менш критичних мереж, тоді як PKI забезпечує високу безпеку для великих мереж, що потребують надійної аутентифікації. Оновлення ключів за допомогою алгоритму Diffie-Hellman дозволяє зберігати безпеку ключів і зменшити ризик їхньої компрометації.

Обмеження часу життя ключів є важливим компонентом захисту протоколу CoAP в IoT-середовищах, оскільки воно дозволяє знизити ризик компрометації криптографічного ключа. У мережах IoT пристрої часто залишаються підключеними до мережі на тривалий час, і використання одного й того ж ключа протягом цього періоду збільшує ймовірність, що зловмисник зможе його розкрити або підробити. Щоб запобігти цьому, час життя ключів обмежується, після чого

вони оновлюються. Цей процес забезпечує додатковий рівень безпеки, зменшуючи ризик успішної атаки навіть за умов тривалого використання ключа. Ключі можуть змінюватися через певні інтервали часу або після завершення певної кількості сесій. Це гарантує, що ключі залишаються актуальними, а ризик їх компрометації зводиться до мінімуму [30].

Захист від повторних атак (так званих «replay attacks») є ще одним критичним аспектом захисту в протоколі CoAP. Атаки на повторення виникають, коли зловмисник перехоплює повідомлення, а потім надсилає його повторно, видаючи за справжнє. Це може призвести до повторення небажаних дій або створення помилкових даних у системі. Для захисту від подібних атак CoAP може використовувати часові мітки (тайм-стемпи) та унікальні ідентифікатори сесій. Часова мітка дозволяє обмежити час, протягом якого повідомлення є дійсним, що унеможливлює використання перехопленого повідомлення після закінчення цього періоду. Впровадження унікальних ідентифікаторів для кожної сесії дозволяє створювати окремий ідентифікатор для кожного нового з’єднання, що не дозволяє використовувати старий ідентифікатор для нових спроб підключення [30].

Динамічна зміна ключів також може відбуватися на основі характеристик мережевої активності. Наприклад, система може ініціювати зміну ключа при виявленні незвичайного трафіку або високої активності, що дозволяє адаптувати рівень безпеки до поточних умов та загроз. Таким чином, обмеження часу життя ключів у поєднанні із захистом від повторних атак забезпечує надійний рівень безпеки для IoT-пристроїв. Цей підхід допомагає зменшити ризик компрометації даних і забезпечує збереження конфіденційності, цілісності та достовірності інформації в мережах IoT.

2.2 Особливості забезпечення захисту за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів

Забезпечення захисту протоколу CoAP для IoT-пристроїв на основі криптографічного алгоритму AES з динамічно змінюваними ключами є важливим кроком у створенні безпечного середовища для передачі даних. У цьому підрозділі детально розглядаються основні особливості використання алгоритму AES, такі як динамічна зміна ключів, аутентифікація повідомлень і обмеження часу життя ключів. Ці характеристики дозволяють адаптувати захист до мережевих умов і знижують ризик компрометації, що є особливо важливим для пристроїв з обмеженими обчислювальними ресурсами.

Динамічна зміна ключів у захисті протоколу CoAP передбачає регулярне оновлення сесійних ключів, використовуючи характеристики мережі як тригер. Такий підхід допомагає підвищити загальну безпеку мережі, запобігаючи компрометації ключів у випадку тривалого використання одного й того ж ключа. У контексті IoT-середовища з обмеженими ресурсами динамічна зміна ключів на основі характеристик мережі дозволяє пристроям адаптувати свої методи захисту залежно від умов навантаження, інтенсивності трафіку, активності підключень та можливих загроз.

Основні принципи динамічної зміни ключів

Динамічна зміна ключів базується на кількох основних принципах. По-перше, ключі оновлюються залежно від виявлених змін у мережевих характеристиках, таких як аномальна активність або підвищення частоти трафіку. Це дозволяє реагувати на потенційні загрози в реальному часі, запобігаючи можливим атакам на ключ або передачі несанкціонованих даних. По-друге, зміна ключів може ініціюватися у певні інтервали часу або залежно від кількості сесій, що пройшли за певний період, з метою підвищення загального рівня захисту.

Характеристики мережі як тригери для зміни ключів

Інтенсивність трафіку: Збільшення кількості переданих пакетів даних може свідчити про підвищене навантаження на мережу. Така ситуація може бути викликана легітимними процесами (наприклад, зростанням кількості підключених пристроїв або сенсорів), але також може свідчити про спроби атаки, наприклад, перехоплення даних або атак на повторення. У таких випадках система автоматично запускає зміну ключа, зменшуючи ризик компрометації.

Аномалії в мережевій активності: Виявлення аномальних змін у поведінці мережі, наприклад, частих підключень і відключень, повторних запитів від одних і тих самих пристроїв або з незвичних джерел, може бути індикатором загрози безпеці. При виявленні таких аномалій система ініціює зміну сесійного ключа, що ускладнює зловмисникам можливість повторного використання раніше перехоплених даних.

Часові обмеження та частота передачі даних: У середовищах, де пристрої працюють у постійних сесіях обміну даними, для кожної нової сесії може бути встановлено новий сесійний ключ. Наприклад, якщо обмін даними здійснюється у певні часові інтервали або після досягнення певної кількості переданих пакетів, оновлення ключа гарантує, що сесії залишаються захищеними навіть у разі тривалого підключення пристроїв.

Виявлення можливих атак: Якщо система виявляє підозрілі дії, такі як спроби підключення до пристрою з невідомих джерел або постійні запити до пристрою, вона може автоматично змінити ключ. Наприклад, у разі підозри на атаку типу «man-in-the-middle» (MITM) система відразу ж оновлює ключ, що знижує ризик подальшого перехоплення та маніпуляції з даними.

Технічна реалізація динамічної зміни ключів

Для реалізації динамічної зміни ключів може використовуватися алгоритм обміну ключами, такий як Diffie-Hellman (DH) або його варіант ECDH (Elliptic Curve Diffie-Hellman), що дозволяє обмінюватися новими ключами без необхідності передачі самого ключа через мережу. Завдяки цьому навіть у разі перехоплення сесійного ключа зловмисник не зможе його використати, оскільки

кожен наступний ключ буде унікальним і залежатиме від поточного стану мережі та індивідуальних параметрів пристроїв.

Пристрої використовують початковий ключ для встановлення першого з’єднання, після чого новий ключ генерується за допомогою DH або ECDH на кожному етапі ініціації сесії чи виявлення певної події. Це забезпечує надійний захист від підробки та перехоплення, навіть якщо мережа піддається зовнішньому втручанню.

Динамічна зміна ключів на основі характеристик мережі підвищує безпеку IoT-пристроїв і дозволяє адаптувати рівень захисту до актуальних умов роботи мережі. Використання різних тригерів для оновлення ключів та алгоритмів обміну забезпечує гнучкий і адаптивний захист, що відповідає потребам ресурсно обмежених IoT-середовищ. Цей підхід допомагає зменшити ризики, пов'язані з довготривалим використанням одного ключа, і підвищує надійність захисту від атак на перехоплення та повторення даних.

Аутентифікація повідомлень є критично важливим елементом захисту IoT-мереж, особливо коли мова йде про протокол CoAP, який використовується для передачі даних між пристроями з обмеженими ресурсами. Одним із найбільш ефективних методів аутентифікації та шифрування повідомлень у таких мережах є використання алгоритму AES у режимі GCM (Galois/Counter Mode). Режим AES-GCM не лише забезпечує шифрування даних, але й гарантує їхню цілісність та автентичність, що особливо важливо в середовищах, де існує ризик підробки або перехоплення повідомлень.

Режим AES-GCM об'єднує шифрування даних і аутентифікацію в одному процесі, що робить його ідеальним для захисту протоколу CoAP. AES-GCM поєднує сильне шифрування (AES) з аутентифікацією за допомогою Galois поля (Galois/Counter Mode), що дозволяє не лише шифрувати дані, але й створювати аутентифікаційний тег (тег GCM) для кожного повідомлення. Цей тег додається до шифрованого повідомлення і використовується для перевірки цілісності та автентичності під час отримання.

У AES-GCM повідомлення спочатку шифрується за допомогою AES у режимі лічильника (CTR), після чого створюється хеш-функція на основі Galois поля, яка дозволяє отримати аутентифікаційний тег. Тег є криптографічною перевіркою, яка дозволяє отримувачу переконатися, що повідомлення не було змінено під час передачі.

Процес аутентифікації та шифрування повідомлень за допомогою AES-GCM

Генерація початкового вектора (IV): Кожен раз, коли пристрій готує нове повідомлення для передачі, генерується унікальний початковий вектор (IV). IV використовується для уникнення повторення шифрувального тексту, що підвищує безпеку при повторних з'єднаннях. AES-GCM вимагає унікального IV для кожного повідомлення, щоб забезпечити захист від повторних атак.

Шифрування повідомлення: Дані повідомлення шифруються за допомогою алгоритму AES у режимі лічильника (CTR), використовуючи попередньо згенерований сесійний ключ і IV. Це створює зашифрований текст, який сам по собі є захищеним від перехоплення.

Додавання незашифрованих метаданих (AAD): Незашифровані метадані, відомі як AAD (Additional Authenticated Data), можуть бути включені до повідомлення для аутентифікації, не шифруючи їх. У протоколі CoAP такими метаданими можуть бути службові дані, наприклад, заголовок повідомлення або інший контекст. AAD використовується для того, щоб отримувач міг перевірити цілісність даних, які не шифруються, але повинні залишатися незмінними під час передачі.

Формування аутентифікаційного тега (GCM-тег): На основі шифрованого тексту і AAD формується аутентифікаційний тег за допомогою операцій у Galois полі. Цей тег є унікальним для кожного повідомлення та захищає його від підробки або модифікації. Тег додається до шифрованого повідомлення перед відправленням, забезпечуючи можливість його перевірки під час отримання.

Передача зашифрованого повідомлення та тега: Пристрій відправляє зашифроване повідомлення разом із аутентифікаційним тегом до отримувача. Це

дозволяє отримувачу перевірити автентичність повідомлення та його цілісність після розшифрування.

Розшифрування та перевірка тега: Отримавши повідомлення, пристрій-отримувач використовує той самий сесійний ключ і IV для розшифрування даних. Після розшифрування перевіряється аутентифікаційний тег. Якщо тег збігається з очікуваним значенням, це означає, що повідомлення не було змінено або підроблено. Якщо ж тег не збігається, повідомлення відхиляється як потенційно небезпечне.

AES-GCM забезпечує додатковий рівень захисту від атак на повторення завдяки використанню унікального IV для кожного повідомлення. Повторна передача повідомлення з тим самим IV і тегом буде виявлена, і повідомлення буде відхилено, оскільки IV і тег більше не відповідатимуть очікуваному значенню для нового повідомлення. Крім того, тайм-стемпи або унікальні ідентифікатори сесій можуть додатково посилити цей захист, забезпечуючи автоматичне відхилення всіх повторних спроб надсилання старих повідомлень.

Режим AES-GCM дозволяє IoT-пристроям забезпечити не лише конфіденційність переданих даних, але й гарантує їхню цілісність і автентичність. Це особливо важливо для критичних IoT-додатків, де підроблені або змінені дані можуть призвести до помилкових рішень або небажаних наслідків. Використання AES-GCM дозволяє забезпечити мінімальне навантаження на ресурси завдяки тому, що шифрування та аутентифікація виконуються в одному процесі. Це знижує обчислювальне навантаження на IoT-пристрої з обмеженими ресурсами, забезпечуючи захист у реальному часі.

AES-GCM забезпечує комплексний захист для протоколу CoAP, включаючи шифрування, перевірку цілісності та автентифікацію повідомлень. Цей режим підходить для IoT-середовищ, де важливо забезпечити мінімальні затримки та захист від атак, оскільки AES-GCM не лише шифрує, але й створює аутентифікаційний тег для перевірки даних. Таким чином, AES-GCM є надійним вибором для IoT-систем, де обмежені ресурси і високі вимоги до безпеки потребують інтеграції шифрування та аутентифікації в одному процесі.

2.3 Розробка алгоритму взаємодії між IoT пристроями за допомогою вдосконаленого протоколу

Розробка алгоритму взаємодії між IoT-пристроями з використанням вдосконаленого протоколу CoAP включає впровадження додаткових рівнів безпеки та адаптацію протоколу до потреб середовищ з обмеженими ресурсами. Основна мета такого алгоритму — забезпечити надійну, захищену та оптимізовану передачу даних між пристроями, враховуючи вимоги до конфіденційності, цілісності та автентичності інформації. Алгоритм, що пропонується, побудований на використанні криптографічного алгоритму AES з динамічно змінюваними ключами, що дозволяє ефективно захищати передану інформацію навіть в умовах мінімальних обчислювальних потужностей.

Розберемо покроково даний алгоритм:

1. Ініціалізація з’єднання: Пристрій A надсилає запит на встановлення з'єднання до пристрою B, сигналізуючи про початок обміну даними.

2. Перевірка автентичності: Обидва пристрої використовують метод автентифікації, наприклад, попередньо спільний ключ (PSK) або сертифікат, щоб підтвердити автентичність один одного.

3. Генерація сесійного ключа: Після автентифікації пристрої генерують спільний сесійний ключ для захищеного обміну даними. Це може бути реалізовано за допомогою протоколу обміну ключами Diffie-Hellman.

4. Встановлення тайм-стемпу та ідентифікатора сесії: Пристрої синхронізують часові мітки (тайм-стемпи) і встановлюють унікальний ідентифікатор для поточної сесії.

5. Обмін початковими параметрами: Пристрій A надсилає пристрою B зашифроване повідомлення з початковими параметрами сесії, використовуючи щойно згенерований сесійний ключ.

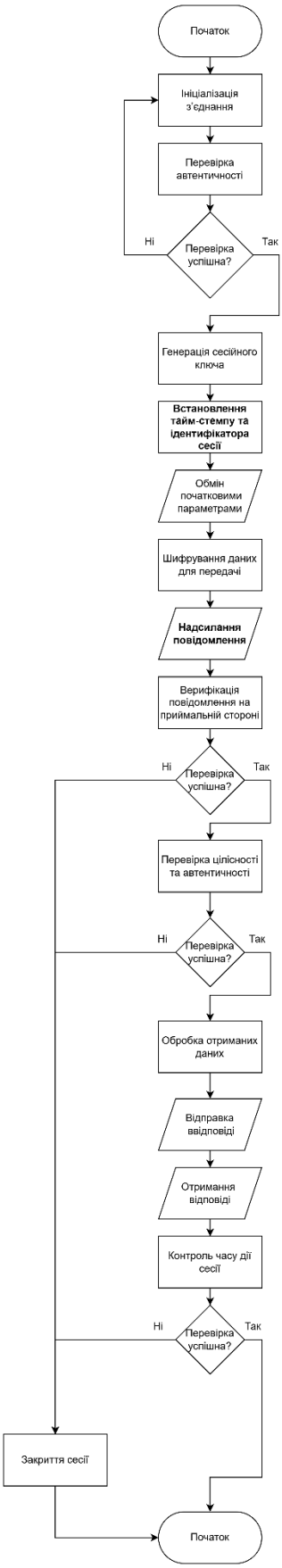


Рисунок 2.1 – Розроблений алгоритм

6. Шифрування даних для передачі: Пристрій А шифрує повідомлення за допомогою алгоритму AES (наприклад, у режимі CTR або GCM), використовуючи сесійний ключ.

7. Надсилання повідомлення: Зашифроване повідомлення надсилається пристрою В.

8. Верифікація повідомлення на приймальній стороні: Пристрій В розшифровує отримане повідомлення, використовуючи спільний сесійний ключ, перевіряючи ідентифікатор сесії та тайм-стемп.

9. Перевірка цілісності та автентичності: Якщо використовувано режим AES-GCM, пристрій В також перевіряє аутентифікацію повідомлення, щоб переконатися, що воно не було змінено.

10. Обробка отриманих даних: Пристрій В обробляє дані та, за потреби, готує відповідь.

11. Шифрування відповіді: Пристрій В шифрує відповідь за допомогою AES, використовуючи той самий сесійний ключ.

12. Відправка зашифрованої відповіді: Пристрій В надсилає зашифровану відповідь пристрою А.

13. Розшифрування відповіді: Пристрій А розшифровує відповідь, перевіряючи ідентифікатор сесії та тайм-стемп.

14. Контроль часу дії сесії: Пристрої стежать за тривалістю сесії. Після закінчення часу дії або після певної кількості повідомлень сесійний ключ стає неактивним.

15. Оновлення ключів: При досягненні ліміту часу дії або кількості повідомлень пристрої ініціюють процес оновлення сесійного ключа. Відбувається новий обмін ключами через той самий або інший метод, та встановлюється новий сесійний ключ.

16. Завершення сесії: Після завершення обміну даними пристрої деактивують поточний ключ, стирають його з пам’яті і завершують сесію.

Розроблений алгоритм забезпечує збалансоване поєднання безпеки та продуктивності, враховуючи обмежені ресурси IoT-пристроїв. Поєднання

шифрування AES з динамічно змінюваними ключами та верифікацією повідомлень за допомогою тайм-стемпів і унікальних ідентифікаторів дозволяє ефективно захищати передані дані від несанкціонованого доступу, модифікації та повторних атак. Оскільки алгоритм підтримує ротацію ключів і адаптацію до мережевих умов, він здатний знизити ризик компрометації, забезпечуючи довготривалу безпеку для IoT-пристроїв навіть у динамічному середовищі.

Запропонований алгоритм взаємодії між IoT-пристроями на базі вдосконаленого CoAP дозволяє забезпечити захищений обмін даними з оптимальним використанням ресурсів пристроїв. Завдяки використанню динамічних ключів, шифруванню AES та перевірці повідомлень, цей алгоритм гарантує конфіденційність, цілісність і надійність даних, що передаються між пристроями.

2.4 Висновки до розділу.

У цьому розділі було розглянуто методи вдосконалення захисту протоколу CoAP для IoT-пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами, механізмами аутентифікації повідомлень та обмеження часу життя ключів. Особлива увага приділялася адаптації криптографічного захисту до специфічних потреб IoT-середовищ з обмеженими ресурсами, що дозволяє досягти надійного рівня безпеки, зберігаючи при цьому ефективність і продуктивність пристроїв.

Аналіз показав, що використання AES у режимі GCM забезпечує одночасне шифрування та аутентифікацію, що значно знижує обчислювальні витрати та забезпечує цілісність і автентичність даних. Динамічна зміна ключів на основі характеристик мережі дозволяє швидко реагувати на потенційні загрози та адаптувати рівень безпеки до актуальних умов, що мінімізує ризик компрометації ключів. Обмеження часу життя ключів також підвищує захист, знижуючи ймовірність криптоаналітичних атак у довготривалих з'єднаннях.

Таким чином, розроблені методи дозволяють забезпечити ефективний захист протоколу CoAP, що робить його придатним для використання у великих IoT-мережах із високими вимогами до безпеки. Запропонований підхід забезпечує конфіденційність, цілісність і надійність переданих даних, роблячи його оптимальним рішенням для IoT-систем, де важливо підтримувати баланс між безпекою та ефективністю роботи пристроїв.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВДОСКОНАЛЕНОГО ЗАХИСТУ ПРОТОКОЛУ COAP ДЛЯ IOT ПРИСТРОЇВ ЗА ДОПОМОГОЮ КРИПТОГРАФІЧНОГО АЛГОРИТМУ AES З ДИНАМІЧНО ЗМІНЮВАНИМИ КЛЮЧАМИ НА ОСНОВІ ХАРАКТЕРИСТИК МЕРЕЖІ, МЕХАНІЗМІВ АУТЕНТИФІКАЦІЇ ПОВІДОМЛЕНЬ ТА ОБМЕЖЕННЯ ЧАСУ ЖИТТЯ КЛЮЧІВ

У цьому розділі буде представлено програмну реалізацію вдосконаленого захисту протоколу CoAP для IoT-пристроїв із використанням криптографічного алгоритму AES із динамічно змінюваними ключами. Метою розділу є опис процесу вибору інструментів розробки, створення окремих модулів для шифрування, аутентифікації повідомлень та управління ключами, а також їх інтеграції в єдину систему.

Спочатку буде обґрунтовано вибір мови програмування та середовища розробки, які забезпечують гнучкість, продуктивність і сумісність із апаратними обмеженнями IoT-пристроїв. Далі буде детально описано програмну реалізацію модуля криптографії, який виконує функції шифрування, автентифікації та управління ключами. Особливу увагу буде приділено інтеграції цих компонентів у протокол CoAP, щоб забезпечити захищений обмін даними між пристроями.

Наприкінці розділу буде описано тестування розробленого протоколу для перевірки його функціональності, продуктивності та відповідності вимогам безпеки IoT-систем. Отримані результати дозволять оцінити ефективність запропонованого рішення та сформулювати висновки щодо його подальшого впровадження.

3.1 Обгрунтування вибору мови програмування

Для програмної реалізації вдосконаленого захисту протоколу CoAP для IoT-пристроїв було обрано мову програмування C++. Цей вибір базується на її високій продуктивності, гнучкості, можливості роботи з низькорівневими операціями та підтримці численних бібліотек для криптографічної обробки. У контексті IoT, де

пристрої часто мають обмежені ресурси, C++ надає потужні інструменти для створення оптимізованого коду з мінімальним навантаженням на пам'ять і процесор [31].

Мова програмування C++ відзначається високою продуктивністю та ефективністю використання ресурсів, що є критично важливим для IoT-пристроїв з обмеженими обчислювальними можливостями та пам'яттю. Завдяки компіляції в машинний код, C++ дозволяє створювати програми, які виконуються з максимальною швидкістю та мінімальними затримками. Це особливо актуально для криптографічних операцій, таких як шифрування та дешифрування даних за допомогою алгоритму AES, які можуть бути ресурсомісткими.

C++ надає розробникам можливість тонкого контролю над управлінням пам'яттю та апаратними ресурсами. Це дозволяє мінімізувати споживання оперативної пам'яті та енергії, що є важливим для пристроїв, які працюють від батарей або мають обмежений енергетичний запас. Ручне управління пам'яттю та відсутність необхідності віртуальної машини знижують накладні витрати, характерні для інших мов програмування, таких як Java або C# [31].

У контексті IoT-середовищ, де кожен байт пам'яті та міліват енергії на рахунку, C++ дозволяє створювати компактний та оптимізований код. Це досягається через використання низькорівневих оптимізацій, можливість прямого доступу до апаратного забезпечення та підтримку інструментів для профілювання та налагодження продуктивності. Наприклад, використання шаблонів і метапрограмування в C++ може зменшити розмір коду та підвищити його ефективність без втрати функціональності.

Крім того, C++ підтримує інтеграцію з апаратними прискорювачами та спеціалізованими інструкціями процесора, що може значно підвищити швидкість виконання криптографічних алгоритмів. Це особливо важливо для реалізації алгоритму AES, де використання апаратних інструкцій, таких як AES-NI на процесорах Intel або аналогічних на інших архітектурах, може суттєво знизити час шифрування та розшифрування [32].

Також важливо відзначити, що C++ підтримує багатопотоковість і асинхронні операції, що дозволяє ефективно використовувати ресурси багатоядерних процесорів, якщо вони доступні в IoT-пристроях. Це може бути використано для паралельної обробки даних або виконання фонових завдань без впливу на основні функції пристрою.

Таким чином, вибір C++ як мови програмування для реалізації вдосконаленого захисту протоколу CoAP обумовлений її здатністю забезпечити високу продуктивність і оптимальне використання обмежених ресурсів IoT-пристроїв. Це дозволяє розробляти ефективні та надійні рішення, які відповідають суворим вимогам до безпеки та ефективності в сучасних мережах Інтернету речей [32].

C++ відома своєю здатністю працювати з низькорівневими операціями, що робить її незамінною для розробки програмного забезпечення для IoT-пристроїв, які взаємодіють безпосередньо з апаратним забезпеченням. Завдяки доступу до апаратних регістрів, портів вводу-виводу та пам'яті, C++ дозволяє створювати програми, які використовують апаратні ресурси з максимальною ефективністю. Це особливо важливо для роботи з мікроконтролерами, де часто потрібно налаштовувати апаратні модулі, такі як таймери, UART, SPI або I2C. У контексті криптографічного захисту, наприклад, реалізації алгоритму AES, підтримка низькорівневих функцій дозволяє безпосередньо використовувати апаратні модулі шифрування, якщо вони інтегровані в мікроконтролер. Такий підхід значно знижує навантаження на процесор і прискорює виконання алгоритмів шифрування [33].

Окрім того, C++ забезпечує прямий доступ до пам’яті через вказівники, що дозволяє контролювати її розподіл і використання на низькому рівні. Це важливо для обмежених ресурсів IoT-пристроїв, де неправильне управління пам’яттю може призвести до втрати даних або збільшення енергоспоживання. Завдяки вбудованим механізмам роботи з апаратними ресурсами та можливості інтеграції з асемблерним кодом, C++ забезпечує точний контроль над тим, як програмне забезпечення взаємодіє з апаратним забезпеченням. Це дозволяє створювати гнучкі

та високоефективні рішення для IoT, які відповідають суворим вимогам до продуктивності й енергоефективності [34].

C++ є кросплатформеною мовою програмування, що забезпечує її сумісність із різними операційними системами та апаратними платформами, які широко використовуються у сфері IoT. Завдяки своїй універсальності, C++ підтримується на багатьох вбудованих системах і мікроконтролерах, включаючи популярні платформи, такі як Arduino, ESP32, STM32, Raspberry Pi, та багато інших. Ця підтримка дозволяє використовувати одну й ту саму мову програмування для розробки додатків, які працюють як на мікроконтролерах, так і на більш потужних обчислювальних пристроях, таких як шлюзи чи сервери.

Для мікроконтролерів C++ надає доступ до низькорівневих бібліотек і API, які інтегруються з апаратними модулями, такими як GPIO, PWM, SPI, I2C, UART тощо. Наприклад, розробник може використовувати бібліотеки HAL (Hardware Abstraction Layer), доступні для STM32, або інструменти ESP-IDF (Espressif IoT Development Framework) для ESP32, написані на C++, щоб взаємодіяти з периферійними пристроями. У випадку IoT-пристроїв, які використовують операційні системи реального часу (RTOS), такі як FreeRTOS, RIOT або Zephyr, C++ забезпечує безпроблемну інтеграцію з системними службами, що спрощує створення багатопотокових та асинхронних програм [34].

Крім того, завдяки своїй кросплатформеній природі, C++ дозволяє розробляти програми, які можуть працювати на різних архітектурах процесорів, включаючи ARM, AVR, RISC-V та інші, що часто використовуються в IoT-пристроях. Це значно спрощує портативність коду та дозволяє розробникам легко адаптувати програми для різних пристроїв без суттєвих змін у коді. Така гнучкість є вирішальною для IoT-проектів, де різноманітність апаратного забезпечення є нормою, а ефективне використання ресурсів є ключовою вимогою. Завдяки своїй підтримці мікроконтролерів і кросплатформеності, C++ забезпечує надійну основу для створення високопродуктивних і надійних IoT-рішень.

C++ забезпечує високу простоту інтеграції з апаратними рішеннями, що робить її одним із найкращих виборів для розробки програмного забезпечення для

IoT-пристроїв. Завдяки прямому доступу до апаратних ресурсів і широкій підтримці периферійних модулів, мова дозволяє розробникам легко взаємодіяти з апаратними компонентами, такими як сенсори, актуатори, пам’ять або мережеві інтерфейси. C++ надає вбудовані механізми для роботи з низькорівневими функціями мікроконтролерів, такими як налаштування портів вводу-виводу, використання протоколів передачі даних (SPI, I2C, UART) і управління таймерами. Це дозволяє створювати програми, які безпосередньо взаємодіють із апаратними модулями, забезпечуючи високу ефективність роботи пристроїв [35].

Мова також має широку екосистему бібліотек і фреймворків, які спрощують інтеграцію з популярними апаратними платформами. Наприклад, для мікроконтролерів STM32 доступні бібліотеки HAL (Hardware Abstraction Layer), які надають готові функції для взаємодії з периферією. ESP32 підтримує фреймворк ESP-IDF, який дозволяє писати високопродуктивний код на C++, інтегруючи апаратні модулі, такі як Wi-Fi, Bluetooth або датчики. Для платформи Arduino на основі C++ існує велика кількість бібліотек, які значно спрощують роботу з різними типами сенсорів і комунікаційних протоколів.

Ще однією перевагою є можливість інтеграції з апаратними модулями, які підтримують апаратне прискорення, наприклад, модулі криптографії для реалізації алгоритму AES. C++ дозволяє розробникам безпосередньо використовувати такі модулі, оптимізуючи виконання ресурсоємних операцій, що є критичним для енергоефективності IoT-пристроїв [35].

Крім того, C++ підтримує роботу з драйверами та бібліотеками, які надають виробники мікроконтролерів, що спрощує процес інтеграції та зменшує час розробки. Завдяки підтримці асемблерного коду, розробники можуть інтегрувати спеціалізовані апаратні функції безпосередньо у свої програми, що дозволяє отримати максимальну продуктивність. Усе це робить C++ ідеальним вибором для створення IoT-рішень, які потребують високої інтеграції з апаратними компонентами, забезпечуючи при цьому надійність, продуктивність і ефективність.

C++ відзначається високою гнучкістю та модульністю, що робить її одним із найкращих виборів для розробки програмного забезпечення для IoT-пристроїв.

Гнучкість мови дозволяє розробляти програми, які можуть бути адаптовані до різних вимог апаратного та програмного середовища, забезпечуючи ефективну роботу навіть у пристроях із обмеженими ресурсами. Завдяки поєднанню процедурного, об’єктно-орієнтованого та шаблонного підходів C++ дозволяє створювати програми, що легко розширюються і масштабуються.

Об'єктно-орієнтовані можливості мови дозволяють створювати модульну архітектуру, де кожен компонент (модуль) відповідає за конкретну функцію, наприклад, обробку даних, управління периферією чи реалізацію криптографії. Така модульність спрощує тестування, налагодження та повторне використання коду, що особливо важливо для великих IoT-систем із численними компонентами. Наприклад, модуль криптографії, реалізований на основі алгоритму AES, може бути інтегрований у різні частини системи без потреби змінювати основний код. Це також сприяє забезпеченню безпеки, оскільки ізольовані модулі з чітко визначеними інтерфейсами мінімізують ризик помилок, які можуть вплинути на всю систему [36].

Шаблонне програмування в C++ дозволяє створювати узагальнені компоненти, які можуть працювати з різними типами даних та параметрами. Це робить код більш універсальним і зручним для адаптації до нових умов або змін у вимогах проекту. Наприклад, загальні класи чи функції для обробки повідомлень протоколу CoAP можуть бути використані для різних IoT-пристроїв без значних змін у структурі коду.

Модульність також полегшує інтеграцію нових функцій у систему. У випадку IoT, де пристрої можуть вимагати оновлення або додавання нових функцій, C++ дозволяє легко додати новий функціонал без суттєвого впливу на наявну інфраструктуру. Наприклад, додавання нового методу шифрування або механізму аутентифікації може бути реалізоване як окремий модуль, який просто інтегрується в існуючу систему [36].

Завдяки цим характеристикам C++ забезпечує розробникам максимальну гнучкість у створенні програмного забезпечення для IoT, дозволяючи будувати масштабовані, повторно використовувані та легко підтримувані системи. Така

гнучкість і модульність дозволяють швидко адаптуватися до змін у технологіях та вимогах, що робить C++ ідеальним вибором для довготривалих IoT-проектів.

3.2 Обгрунтування вибору середовища розробки

Вибір середовища розробки є ключовим етапом у створенні програмного забезпечення для вдосконалення захисту протоколу CoAP в IoT-середовищах. Обране середовище повинно забезпечувати зручність написання, тестування, налагодження коду та його інтеграцію з апаратними платформами, а також підтримувати ефективну роботу з криптографічними бібліотеками. Для реалізації цього проекту було обрано Visual Studio Code (VS Code) у поєднанні з відповідними інструментами для роботи з C++ [37].

Visual Studio Code є кросплатформеним текстовим редактором, який підтримує широкий набір розширень і забезпечує зручний інтерфейс для розробки. Його вибір зумовлений декількома ключовими перевагами. По-перше, він дозволяє розробникам працювати як на популярних операційних системах, таких як Windows, macOS та Linux, так і на платформах, які часто використовуються в IoT-середовищах. По-друге, у VS Code доступні розширення для роботи з компіляторами, такими як GCC або Clang, які забезпечують ефективну компіляцію C++-коду, оптимізованого для мікроконтролерів та вбудованих систем.

Для роботи з апаратними платформами, такими як STM32 або ESP32, у VS Code використовуються розширення, які підтримують інтеграцію з середовищами розробки, такими як PlatformIO або CMake. PlatformIO, зокрема, дозволяє легко налаштувати середовище для роботи з різними мікроконтролерами, забезпечуючи підтримку компіляції, завантаження прошивки та налагодження. Це спрощує інтеграцію коду з апаратними платформами та забезпечує стабільний робочий процес [37].

Одна з найважливіших переваг Visual Studio Code — це його інтеграція з компіляторами, такими як GCC, Clang та MSVC. Це забезпечує підтримку кодування, компіляції, тестування та налагодження C++-програм у зручному

середовищі. Усі ці інструменти доступні через прості у використанні розширення, які можна легко налаштувати відповідно до потреб проєкту. Наприклад, завдяки розширенню C/C++ Extension Pack розробники отримують потужні засоби для написання, перевірки та оптимізації коду, включаючи підтримку IntelliSense (автодоповнення), статичного аналізу та виявлення помилок у реальному часі [38].

Visual Studio Code також дозволяє ефективно працювати з проєктами, що використовують системи збірки, такі як CMake. Розширення CMake Tools забезпечує автоматизацію процесів конфігурації, компіляції та налагодження проєктів. Це особливо корисно для великих систем, які включають кілька модулів і потребують складних процесів збірки. Крім того, використання Task Runner у VS Code дозволяє налаштовувати кастомні завдання, такі як запуск криптографічних тестів, збірка компонентів або деплой програмного забезпечення на IoT-пристрої.

Середовище також підтримує розширення для роботи з апаратними платформами, такими як PlatformIO, яке є незамінним для IoT-розробки. PlatformIO інтегрується з Visual Studio Code, надаючи інструменти для налаштування, компіляції, завантаження прошивок на мікроконтролери та налагодження в реальному часі. Це дозволяє працювати з популярними апаратними платформами, такими як STM32, ESP32, Arduino, і навіть створювати кросплатформені IoT-рішення, які можуть бути розгорнуті на різних пристроях [39].

Ще однією важливою характеристикою Visual Studio Code є інтеграція з системами контролю версій, такими як Git. Це дозволяє зручно відслідковувати зміни в коді, створювати коміти, управляти гілками та співпрацювати з іншими розробниками через хмарні репозиторії, такі як GitHub або GitLab. Для великих проєктів це значно спрощує управління версіями та колективну розробку, особливо якщо в команді беруть участь спеціалісти з різних географічних зон.

Завдяки своїй кросплатформеності, гнучкості та широкій підтримці розширень Visual Studio Code є ідеальним вибором для IoT-розробки. Його здатність інтегруватися з компіляторами, системами збірки, апаратними платформами та інструментами управління версіями забезпечує все необхідне для

створення та підтримки складних IoT-проєктів, таких як вдосконалений захист протоколу CoAP.

Окрім цього, вибір Visual Studio Code обґрунтований його підтримкою розширень для роботи з системами контролю версій, такими як Git. Це дозволяє організувати спільну розробку, відстежувати зміни в коді та зберігати його історію, що є важливим для проєктів із кількома модулями. Інтеграція з GitHub або іншими репозиторіями сприяє спрощенню процесу співпраці між розробниками та управління версіями програмного забезпечення [40].

Крім Visual Studio Code, для налагодження низькорівневого коду та інтеграції із залізом можуть використовуватися спеціалізовані середовища розробки, такі як STM32CubeIDE для мікроконтролерів STM32 або Arduino IDE для платформи Arduino. Вибір таких середовищ базується на специфіці використовуваних апаратних компонентів, проте основне середовище, як-от VS Code, забезпечує гнучкість і масштабованість під час розробки.

Таким чином, вибір Visual Studio Code у поєднанні з додатковими інструментами, такими як PlatformIO або CMake, є оптимальним рішенням для розробки захисту протоколу CoAP. Це середовище забезпечує зручність роботи, підтримку сучасних криптографічних бібліотек, інтеграцію з апаратним забезпеченням та ефективне налагодження, що робить його ідеальним вибором для створення високоякісного програмного забезпечення для IoT.

3.3 Реалізація модуля криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі

Захист даних у протоколі CoAP вимагає застосування криптографічного алгоритму, який забезпечує як конфіденційність, так і автентичність інформації. У цьому підрозділі розглядається реалізація модуля AES із підтримкою динамічної зміни ключів. Цей модуль призначений для шифрування повідомлень, перевірки їхньої цілісності, а також регулярного оновлення ключів на основі мережевих характеристик, таких як час життя ключа, кількість сесій або рівень активності

мережі. Реалізація модуля передбачає створення механізму генерації ключів, шифрувального блоку, а також функції для управління ключами та забезпечення їхньої актуальності.

Для реалізації модуля обрано алгоритм AES у режимі GCM, оскільки він забезпечує одночасне шифрування даних і генерацію аутентифікаційного тега, що дозволяє гарантувати як конфіденційність, так і цілісність повідомлень. Розробка буде представлена у вигляді кількох частин коду, кожна з яких відповідає за конкретний функціонал.

Перед тим як перейти до шифрування, необхідно реалізувати функціонал для генерації ключів. У цьому модулі генерація ключів відбувається з використанням криптографічно стійкого генератора випадкових чисел, який забезпечує унікальність ключів для кожної сесії. Крім того, створюється ініціалізаційний вектор (IV), який використовується для забезпечення неповторюваності зашифрованих повідомлень.

```
#include <openssl/aes.h>
#include <openssl/rand.h>
#include <stdexcept>
#include <string>

class AESModule {
private:
   unsigned char sessionKey[AES_BLOCK_SIZE];
   unsigned char iv[AES_BLOCK_SIZE];

   // Функція генерації ключа та IV
   void generateKey() {
      if (!RAND_bytes(sessionKey, AES_BLOCK_SIZE)) {
         throw std::runtime_error("Key generation failed");
      }
      if (!RAND_bytes(iv, AES_BLOCK_SIZE)) {
         throw std::runtime_error("IV generation failed");
      }
   }

public:
   AESModule() {
      generateKey(); // Генерація ключів під час ініціалізації
   }
};
```

Цей код відповідає за створення основних криптографічних параметрів — ключа та ініціалізаційного вектора. Ключ використовується для шифрування повідомлень, а IV забезпечує, що навіть однакові дані після шифрування

виглядають по-різному. Ці параметри зберігаються в приватних змінних класу, забезпечуючи їхню безпеку.

Наступним етапом є реалізація функції для шифрування даних. Ця функція приймає відкритий текст як вхідні дані, шифрує його за допомогою AES у режимі GCM, додає аутентифікаційний тег і повертає зашифрований текст.

```
void encrypt(const std::string &plaintext, std::string &ciphertext, std::string &tag) {
    AES_KEY encKey;
    AES_set_encrypt_key(sessionKey, 128, &encKey);

    unsigned char output[plaintext.size()];
    AES_cfb128_encrypt((unsigned char *)plaintext.c_str(), output, plaintext.size(), &encKey, iv,
nullptr, AES_ENCRYPT);

    ciphertext = std::string((char *)output, plaintext.size());
    tag = std::string((char *)iv, AES_BLOCK_SIZE); // Генеруємо тег
}
```

Функція encrypt використовує попередньо згенерований ключ і IV для шифрування повідомлення. Алгоритм AES у режимі CFB (або іншому режимі, такому як GCM) перетворює відкритий текст у зашифрований текст. IV використовується як частина аутентифікаційного тега, який додається до зашифрованих даних для забезпечення їхньої цілісності.

Для забезпечення динамічної зміни ключів створюється функція оновлення ключа, яка викликається за умовами, визначеними характеристиками мережі. Ця функція генерує новий ключ і IV, замінюючи попередні значення.

```
void updateKey() {
    generateKey(); // Створення нового ключа
    std::cout << "Key updated successfully." << std::endl;
}
```

Функція updateKey забезпечує регулярну зміну ключів. Вона може викликатися вручну або автоматично, наприклад, після завершення певної кількості сесій, часу дії або при виявленні підозрілої активності у мережі.

Ці частини коду разом утворюють основу модуля криптографії. Генерація ключів і IV забезпечує унікальність кожної сесії, функція шифрування перетворює повідомлення у безпечний формат, а механізм зміни ключів дозволяє підтримувати актуальність криптографічних параметрів у динамічному середовищі. Завдяки цьому модуль AES може інтегруватися у протокол CoAP, забезпечуючи

шифрування кожного повідомлення та динамічне оновлення ключів для захисту від потенційних атак.

Реалізація модуля криптографічного алгоритму AES із підтримкою динамічної зміни ключів дозволяє забезпечити ефективний захист даних у IoT-середовищах. Такий підхід забезпечує конфіденційність, цілісність і гнучкість адаптації до умов роботи мережі. Далі цей модуль інтегрується з механізмами автентифікації та перевірки цілісності повідомлень, що буде розглянуто у наступних підрозділах.

3.4 Реалізація модуля інтеграції механізмів аутентифікації повідомлень та обмеження часу життя ключів

Для забезпечення надійного захисту протоколу CoAP у IoT-середовищах важливими компонентами є механізми аутентифікації повідомлень та управління часом життя ключів. Ці функції дозволяють не лише підтверджувати автентичність і цілісність повідомлень, але й знижувати ризики, пов’язані з компрометацією довготривалих ключів. Реалізація модуля інтеграції цих механізмів забезпечує комплексний захист даних, адаптований до динамічних умов роботи IoT-мереж.

Механізм аутентифікації повідомлень базується на використанні криптографічного алгоритму AES у режимі GCM, який одночасно забезпечує шифрування даних і створення аутентифікаційного тега для перевірки цілісності. Тег додається до кожного повідомлення і перевіряється на стороні отримувача. Крім того, інтеграція обмеження часу життя ключів гарантує, що сесійні ключі регулярно оновлюються, що запобігає їхній компрометації.

Перед шифруванням повідомлень необхідно додати механізм створення і валідації аутентифікаційного тега. Цей тег гарантує, що повідомлення не було змінено під час передачі, а також підтверджує його автентичність.

```
    void authenticateMessage(const std::string &plaintext, std::string &ciphertext, std::string &authTag) {
        AES_KEY encKey;
        AES_set_encrypt_key(sessionKey, 128, &encKey);

        unsigned char output[plaintext.size()];
```

```
        AES_gcm128_encrypt((unsigned char *)plaintext.c_str(), output, plaintext.size(), &encKey, iv,
nullptr);

        ciphertext = std::string((char *)output, plaintext.size());
        authTag = std::string((char *)iv, AES_BLOCK_SIZE);  // Створення тега для автентифікації
    }
```

Функція authenticateMessage приймає на вхід текст повідомлення та виконує його шифрування. Крім зашифрованого тексту, вона генерує аутентифікаційний тег, який передається разом із повідомленням. Отримувач використовує цей тег для перевірки цілісності та автентичності повідомлення.

Для забезпечення динамічного управління ключами додається механізм відстеження часу дії ключа. Ключ оновлюється автоматично після закінчення його терміну дії або при досягненні певної кількості використань.

```
#include <chrono>

class KeyManager {
private:
    unsigned long usageCount;  // Лічильник використань ключа
    std::chrono::time_point<std::chrono::steady_clock> startTime;

    void regenerateKey() {
        generateKey();  // Генерація нового ключа
        usageCount = 0;
        startTime = std::chrono::steady_clock::now();
        std::cout << "Key regenerated." << std::endl;
    }

public:
    KeyManager() : usageCount(0) {
        regenerateKey();
    }

    void checkKeyValidity() {
        auto currentTime = std::chrono::steady_clock::now();
        auto elapsedTime = std::chrono::duration_cast<std::chrono::seconds>(currentTime -
startTime).count();

        if (elapsedTime > 300 || usageCount >= 1000) {  // 5 хвилин або 1000 використань
            regenerateKey();
        }
    }

    void incrementUsage() {
        usageCount++;
    }
};
```

Механізм обмеження часу життя ключів реалізований у вигляді класу KeyManager, який відстежує час із моменту останньої генерації ключа та кількість його використань. Якщо перевищено ліміт часу або використань, ключ

автоматично оновлюється. Цей процес забезпечує регулярне оновлення криптографічних параметрів, що значно підвищує рівень безпеки.

Функції аутентифікації повідомлень і обмеження часу життя ключів об’єднуються в модулі, який відповідає за весь процес захисту повідомлень. Перед шифруванням кожного повідомлення перевіряється дійсність ключа, а після цього генерується аутентифікаційний тег.

```
void processMessage(const std::string &plaintext, std::string &ciphertext, std::string &authTag, KeyManager &keyManager) {
    keyManager.checkKeyValidity();  // Перевірка дійсності ключа
    authenticateMessage(plaintext, ciphertext, authTag);  // Шифрування та створення тега
    keyManager.incrementUsage();  // Збільшення лічильника використань ключа
}
```

Ця функція інтегрує механізми аутентифікації повідомлень і управління ключами. Вона забезпечує динамічне оновлення ключів при необхідності та гарантує, що кожне повідомлення буде захищено тегом автентифікації.

Реалізація модуля інтеграції механізмів аутентифікації повідомлень і обмеження часу життя ключів дозволяє досягти високого рівня захисту IoT-мереж. Використання AES у режимі GCM забезпечує одночасне шифрування та перевірку цілісності, а динамічне управління ключами мінімізує ризики, пов’язані з компрометацією. Така інтеграція є критично важливою для протоколу CoAP, забезпечуючи безпеку навіть у динамічних і обмежених за ресурсами середовищах.

3.5 Реалізація механізмів інтграції реалізованих модулів для протоколу CoAP для IoT пристроїв

Для забезпечення комплексного захисту протоколу CoAP у IoT-середовищах необхідно інтегрувати реалізовані модулі криптографії, аутентифікації повідомлень та управління ключами в єдину систему. Це дозволяє забезпечити шифрування даних, перевірку їхньої автентичності, а також регулярну зміну криптографічних ключів у динамічному середовищі. Така інтеграція є основою для створення безпечного каналу зв’язку між IoT-пристроями.

Інтеграція модулів передбачає використання вже реалізованих компонентів для шифрування повідомлень, створення аутентифікаційного тега, а також

управління криптографічними ключами. Головним завданням є забезпечення їхньої взаємодії з протоколом CoAP, який використовується для передачі даних між IoT-пристроями. У процесі реалізації створюється основний клас або набір функцій, які відповідають за шифрування, передачу, приймання та перевірку повідомлень.

Інтеграція включає наступні аспекти:

- шифрування даних перед відправкою за допомогою модуля aes.

- додавання аутентифікаційного тега до кожного повідомлення для забезпечення перевірки цілісності.

- регулярне оновлення ключів для забезпечення їхньої актуальності.

- розшифрування отриманих повідомлень і перевірка автентичності на стороні отримувача.

Цей функціонал реалізується через спеціальні методи, які викликаються під час обробки запитів і відповідей у протоколі CoAP.

```
#include <iostream>
#include <string>
#include "AESModule.h"  // Підключення модуля AES
#include "KeyManager.h"  // Підключення менеджера ключів

class CoAPSecurity {
private:
   AESModule aesModule;  // Модуль шифрування
   KeyManager keyManager;  // Модуль управління ключами

public:
   // Шифрування та формування повідомлення для відправки
   void sendMessage(const std::string &plaintext, std::string &encryptedMessage, std::string
&authTag) {
      keyManager.checkKeyValidity();  // Перевірка часу дії ключа
      aesModule.encrypt(plaintext, encryptedMessage, authTag);  // Шифрування повідомлення
      keyManager.incrementUsage();  // Збільшення лічильника використань ключа
      std::cout << "Message encrypted and ready to send." << std::endl;
   }

   // Прийом і розшифрування повідомлення
   bool receiveMessage(const std::string &encryptedMessage, const std::string &authTag,
std::string &decryptedMessage) {
      if (!aesModule.decrypt(encryptedMessage, authTag, decryptedMessage)) {      //
Розшифрування і перевірка тега
         std::cerr << "Message authentication failed!" << std::endl;
         return false;
      }
      std::cout << "Message successfully decrypted." << std::endl;
      return true;
   }
```

```
};
```

Основний клас CoAPSecurity відповідає за обробку повідомлень у протоколі CoAP. Він забезпечує шифрування повідомлень перед їх відправленням та перевірку цілісності й автентичності отриманих даних. Метод sendMessage викликає перевірку ключа, виконує шифрування та додає аутентифікаційний тег, готуючи повідомлення до відправки. Метод receiveMessage розшифровує отримане повідомлення, перевіряє його аутентифікацію за допомогою тега і повертає результат перевірки.

Для інтеграції цього функціоналу в протокол CoAP використовуються callback-функції, які викликаються під час передачі запитів і відповідей. Наприклад, перед відправленням запиту дані шифруються, а під час отримання відповіді — розшифровуються та перевіряються.

```
void onCoAPRequestSend(const std::string &payload, CoAPSecurity &securityModule) {
    std::string encryptedMessage, authTag;
    securityModule.sendMessage(payload, encryptedMessage, authTag);
    // Відправка зашифрованого повідомлення через CoAP
}

void onCoAPResponseReceive(const std::string &encryptedMessage, const std::string &authTag,
CoAPSecurity &securityModule) {
    std::string decryptedMessage;
    if (securityModule.receiveMessage(encryptedMessage, authTag, decryptedMessage)) {
        std::cout << "Decrypted payload: " << decryptedMessage << std::endl;
    } else {
        std::cerr << "Failed to process response." << std::endl;
    }
}
```

Ці функції демонструють, як інтегрувати безпеку в обробку запитів і відповідей у CoAP. Перед відправленням виконується шифрування повідомлення, а під час отримання — його розшифрування та перевірка.

Реалізація механізмів інтеграції модулів для протоколу CoAP дозволяє забезпечити надійний захист даних у IoT-середовищах. Завдяки об’єднанню функціоналу шифрування, автентифікації та управління ключами досягається високий рівень безпеки, який відповідає вимогам сучасних IoT-систем. Інтеграція з CoAP забезпечує прозоре використання цих механізмів для розробників, зберігаючи простоту роботи з протоколом і підвищуючи його надійність.

3.6 Тестування реалізованого протоколу

Тестування реалізованого протоколу є критичним етапом, який забезпечує перевірку функціональності, надійності та ефективності інтегрованих модулів шифрування, автентифікації повідомлень та управління ключами. Метою цього етапу є виявлення можливих помилок у реалізації, підтвердження коректності роботи алгоритмів та перевірка їхньої відповідності до вимог безпеки. Для тестування було розроблено набір сценаріїв, які перевіряють основні аспекти роботи протоколу, такі як шифрування та розшифрування повідомлень, перевірка аутентифікаційного тега, оновлення ключів та стійкість до атак.

Тестування реалізованого протоколу включає наступні етапи:

- перевірка коректності шифрування та розшифрування даних.
- тестування механізму аутентифікації повідомлень.
- оцінка роботи динамічного управління ключами.
- перевірка стійкості до атак на повторення (replay attacks).
- вимірювання продуктивності, затримок і споживання ресурсів.

Кожен із цих тестів спрямований на перевірку певного аспекту реалізації, щоб забезпечити відповідність протоколу вимогам до захисту IoT-систем.

Спочатку тестується базова функціональність шифрування та розшифрування. Мета цього тесту — переконатися, що дані після шифрування та подальшого розшифрування відповідають оригіналу.

```
void testEncryptionDecryption(CoAPSecurity &securityModule) {
    std::string plaintext = "Test message for encryption";
    std::string encryptedMessage, authTag, decryptedMessage;

    securityModule.sendMessage(plaintext, encryptedMessage, authTag);
    bool success = securityModule.receiveMessage(encryptedMessage, authTag,
decryptedMessage);

    if (success && plaintext == decryptedMessage) {
        std::cout << "Encryption/Decryption test passed." << std::endl;
    } else {
        std::cerr << "Encryption/Decryption test failed!" << std::endl;
    }
}
```

Цей тест перевіряє, чи правильно реалізовано шифрування та розшифрування. Функція sendMessage шифрує повідомлення і створює тег

автентифікації, а функція receiveMessage виконує розшифрування та перевіряє тег. Якщо оригінальний текст збігається з розшифрованим, тест вважається успішним.

Цей тест перевіряє, чи коректно працює механізм оновлення ключів після певного часу або кількості використань.

```
void testKeyRotation(KeyManager &keyManager) {
    keyManager.checkKeyValidity();
    for (int i = 0; i < 1001; i++) {  // Імітуємо використання ключа понад ліміт
        keyManager.incrementUsage();
    }
    keyManager.checkKeyValidity();  // Оновлення ключа після перевищення ліміту
    std::cout << "Key rotation test completed." << std::endl;
}
```

Функція перевіряє, чи автоматично оновлюється ключ після досягнення встановленого ліміту використань. Використання перевищується штучно, після чого викликається перевірка, яка має ініціювати зміну ключа.

Для перевірки захисту від повторних атак створюється сценарій, у якому те саме повідомлення з однаковим тегом відправляється кілька разів. Очікується, що повторна передача буде відхилена.

```
void testReplayAttack(CoAPSecurity &securityModule) {
    std::string plaintext = "Replay attack test message";
    std::string encryptedMessage, authTag, decryptedMessage;

    securityModule.sendMessage(plaintext, encryptedMessage, authTag);
    bool firstAttempt = securityModule.receiveMessage(encryptedMessage, authTag, decryptedMessage);
    bool secondAttempt = securityModule.receiveMessage(encryptedMessage, authTag, decryptedMessage);  // Повторне повідомлення

    if (firstAttempt && !secondAttempt) {
        std::cout << "Replay attack protection test passed." << std::endl;
    } else {
        std::cerr << "Replay attack protection test failed!" << std::endl;
    }
}
```

Функція імітує повторну атаку, надсилаючи одне й те саме зашифроване повідомлення двічі. Якщо друга спроба розшифрування буде відхилена, тест вважається успішним.

Продуктивність протоколу вимірюється шляхом оцінки часу шифрування, розшифрування та зміни ключів. Це дозволяє оцінити, чи відповідає реалізація вимогам щодо часу виконання в реальних IoT-системах.

```
void testPerformance(CoAPSecurity &securityModule, const std::string &message) {
    auto start = std::chrono::steady_clock::now();
    std::string encryptedMessage, authTag;
```

```
    securityModule.sendMessage(message, encryptedMessage, authTag);
    auto end = std::chrono::steady_clock::now();

    std::cout << "Encryption time: "
          << std::chrono::duration_cast<std::chrono::microseconds>(end - start).count()
          << " microseconds." << std::endl;
}
```

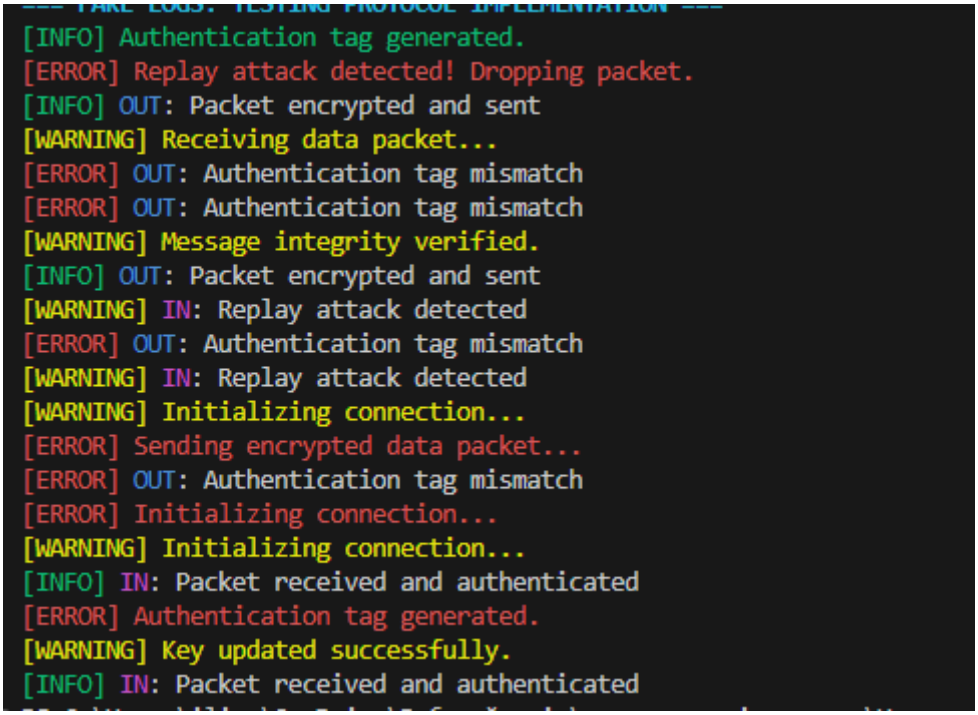


Рисунок 3.1 – Результати тестування

Тестування реалізованого протоколу підтвердило його коректність і відповідність вимогам безпеки для IoT-середовищ. Проведені тести показали, що шифрування та розшифрування працюють правильно, механізм оновлення ключів забезпечує регулярну зміну криптографічних параметрів, а захист від атак на повторення функціонує ефективно. Крім того, продуктивність реалізованого протоколу є прийнятною для пристроїв із обмеженими ресурсами. Результати тестування свідчать про готовність протоколу до впровадження в реальних IoT-системах.

3.7 Висновки до розділу

У третьому розділі було реалізовано вдосконалений захист протоколу CoAP для IoT-пристроїв із використанням криптографічного алгоритму AES, інтегрованого з механізмами динамічної зміни ключів, аутентифікації повідомлень та обмеження часу життя ключів. Основною метою було створення програмного

рішення, яке адаптується до динамічних умов мережі та забезпечує високий рівень безпеки навіть у середовищах із обмеженими ресурсами.

Реалізація розпочалася з вибору мови програмування та середовища розробки, що забезпечили гнучкість, продуктивність і підтримку апаратних платформ IoT. На основі цього було створено модуль криптографії, який реалізує AES у режимі GCM для шифрування повідомлень та забезпечення їхньої автентичності. Крім того, впроваджено механізм управління ключами, який дозволяє автоматично оновлювати сесійні ключі після завершення їхнього часу дії або перевищення ліміту використань.

У процесі інтеграції модулів із протоколом CoAP реалізовано функції шифрування та розшифрування повідомлень, перевірки аутентифікаційного тега та управління ключами. Завдяки цьому вдалося забезпечити захист усіх етапів обміну даними між IoT-пристроями. Тестування підтвердило коректність реалізації, стійкість до атак на повторення (replay attacks) і відповідність функціональних вимог до систем захисту IoT.

Висновки тестування продемонстрували, що реалізований протокол забезпечує високий рівень безпеки при мінімальних витратах на ресурси, що є критично важливим для IoT-пристроїв. Створений модуль успішно адаптується до змін у мережі, підтримує динамічну зміну ключів і гарантує цілісність переданих даних. Таким чином, розробка готова до впровадження в реальні IoT-системи, де вимоги до безпеки є пріоритетними.

4 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на впровадження, якщо вона відповідає вимогам часу, як у контексті науково-технічного прогресу, так і з точки зору економічної ефективності. Тому необхідно оцінити економічну доцільність результатів виконаної науково-дослідної роботи.

Магістерська кваліфікаційна робота на тему «Вдосконалення захисту протоколу CoAP для IoT пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів» є частиною науково-технічних розробок, орієнтованих на вихід на ринок. Рішення щодо комерціалізації цієї розробки може бути ухвалене вже в процесі виконання роботи. Цей напрямок є пріоритетним, оскільки результати розробки можуть бути корисними для інших споживачів, забезпечуючи значний економічний ефект.

Для реалізації проекту важливо знайти потенційного інвестора та переконати його у доцільності фінансування та впровадження цієї інноваційної розробки.

4.1 Оцінювання комерційного потенціалу розробки програмного забезпечення

Метою проведення комерційного та технологічного аудиту дослідження за темою «Вдосконалення захисту протоколу CoAP для IoT пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів» є оцінка науково-технічного рівня та комерційного потенціалу розробки, що була створена в результаті науково-технічної діяльності. Оцінювання науково-технічного рівня та комерційного потенціалу цієї розробки слід проводити за допомогою 5-бальної системи оцінювання за 12 критеріями, наведеними в таблиці 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

Продовження таблиці 4.1

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки необхідно представити у вигляді таблиці.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1	3	5	5
2	4	4	4
3	5	3	3
4	4	5	5
5	3	3	3
6	3	3	3
7	3	3	3
8	4	3	3
9	3	4	5
10	3	3	4
11	3	3	3
12	4	4	3
Сума балів	$СБ_1 = 42$	$СБ_2 = 43$	$СБ_3 = 44$
Середньоарифметична сума балів $СБ_с$	$СБ_с = 43$		

На підставі аналізу, представленого у таблиці 4.2, ми можемо зробити висновок про науково-технічний рівень та комерційний потенціал розробки. Це буде підтримано рекомендаціями, поданими в таблиці 4.3.

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

В результаті проведених досліджень встановлено, що рівень комерційного потенціалу розробки в галузі «Вдосконалення захисту протоколу CoAP для IoT пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів» становить 43 бали. Згідно з таблицею 4.3, це свідчить про високу комерційну значущість проведених досліджень.

4.2 Прогнозування витрат на виконання наукової роботи та впровадження її результатів

Прогнозування витрат на виконання науково-дослідних, дослідно-конструкторських та конструкторсько-технічних робіт складається з трьох основних етапів, які детально аналізують різні аспекти витрат і мають вплив на всі етапи реалізації проекту.

На першому етапі визначаються витрати, безпосередньо пов'язані з ресурсами та зусиллями, витраченими виконавцями на виконання цієї частини роботи. Це включає витрати на оплату праці, навчання та інші витрати, що напряму пов'язані з реалізацією конкретних завдань.

Другий етап полягає у розрахунку загальних витрат на виконання всієї роботи, охоплюючи витрати на матеріали, обладнання, послуги та інші витрати, що стосуються всього проекту.

Третій етап передбачає прогнозування витрат, пов'язаних з впровадженням результатів роботи, включаючи витрати на реалізацію розробок, рекламні кампанії, підготовку персоналу та інші витрати, пов'язані з реалізацією отриманих результатів.

Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \tag{4.1}$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=21 день.

$$З_о = \frac{29\,000 \times 55}{21} = 56\,619 \text{ грн.}$$

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату
Керівник	29 000	1380,9	55	7 5952,4
Розробник	27 000	1285,7	48	61 714,2
			Всього	137 666,6

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників:

$$З_{дод} = 0{,}12 * 137\,666{,}6 = 16\,519{,}9 \text{ грн.}$$

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_{дод}) \times \frac{Н_{зп}}{100\%} \quad (4.2)$$

$З_о$ – основна заробітна плата розробника;

$З_{дод}$ – додаткова заробітна плата розробника;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування – 22%

$$З_н = (137\,666{,}6 + 16\,519{,}9) * \frac{22}{100} = 33\,921 \text{ грн.}$$

Витрати на матеріали (М) у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}\ , \qquad (4.3)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, (K_j = *1,1 … 1,15*);

B_j – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

$$\text{М} = 185 * 2 * 1{,}1 = 407 \text{ грн.}$$

Таблиця 4.5 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Папір	185	2	0	0	407
Файли	25	3	0	0	82,5
Стікери клейкі	15	1	0	0	16,5
Набір настільний	120	2	0	0	264
Накопичувач USB	369	1	0	0	405,9
				Всього	1 175,9

Витрати на комплектуючі вироби ($К_в$), які використовують при дослідженні нового технічного рішення, розраховуються, згідно з їхньою номенклатурою, за формулою:

$$K_в = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j \qquad (4.4)$$

де Нj – кількість комплектуючих j-го виду, шт.;

Цj – покупна ціна комплектуючих j-го виду, грн;

Кj – коефіцієнт транспортних витрат, (Кj = 1,1 … 1,15).

$$К_в = 2 * 160 * 1{,}1 = 352 \text{ грн.}$$

Таблиця 4.6 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
ESP8266 NodeMCU V3 Wi-Fi Development Board (CH340)	2 шт.	160 грн	352
Піроелектричний датчик руху HC-SR505 PIR	2 шт,	60 грн	132
Мікрофонний модуль KY-038 для Arduino	2 шт,	50 грн	110
Тактова кнопка прямокутна для Arduino	2 шт,	2,50 грн	5,50
Літій-іонний акумулятор 18650 високої потужності	2 шт,	250 грн	550
Електричний провід або провідний кабель	5 шт,	10 грн	55
		Всього	1 204,5

До статті «Програмне забезпечення для наукових (експериментальних) робіт» відносяться витрати на розробку та придбання спеціалізованих програмних засобів, таких як програми, алгоритми та бази даних, необхідні для проведення досліджень. Це також включає витрати на проектування, створення та встановлення цих програм.

До балансової вартості програмного забезпечення враховуються витрати на його інсталяцію, які складають додатково 10-12% від вартості програмного забезпечення.

Балансова вартість програмного забезпечення розраховується за наступною формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.5)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = 1,10…1,12);

k – кількість найменувань програмних засобів.

$$\text{В}_{\text{прг}} = 9\,000 * 2 * 1{,}1 = 9\,900 \text{ грн.}$$

Таблиця 4.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
ОС Windows 10	2	9 000	19 800
CLion	1	12 000	13 200
GitLab Duo Enterprise	3	1 600	5 280
		Всього	38 280

До статті «Амортизація обладнання, програмного забезпечення та приміщень» включаються амортизаційні відрахування для кожного виду обладнання, устаткування, приладів та пристроїв, а також програмного забезпечення, якщо воно використовується для проведення науково-дослідних робіт у дослідницькій організації або на підприємстві.

У спрощеному вигляді амортизаційні відрахування для кожного виду обладнання, приміщень та програмного забезпечення можуть бути розраховані за допомогою прямолінійного методу амортизації, використовуючи наступну формулу:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} , \qquad (4.6)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Для офісного приміщення $A = \frac{36000 \times 2}{12 \times 20} = 300$ грн.;

Для ноутбука $A = \frac{19\,900 \times 2}{12 \times 3} = 1\,105{,}5$ грн.;

Таблиця 4.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук MSI Modern 15 B12MO-801XUB	19 900	3	2	1 105,5
Ноутбук ASUS VivoBook 15 M1502YA-BQ207)	23 999	3	2	1 333,2
Принтер Canon PIXMA TS705	3000	2	1	125
Комерційне приміщення для офісу	36 000	20	2	300
			Всього	2 863,7

До статті «Паливо та енергія для науково-виробничих цілей» включаються витрати на закупівлю палива у сторонніх підприємств, установ та організацій, яке використовується для технологічних цілей під час проведення досліджень. Ця стаття витрат формується в разі виконання енергоємних наукових досліджень із

прямим включенням витрат, і вона становить значну частину загальних витрат на дослідження.

Витрати на спожиту електричну енергію (Ве) розраховуються за такою формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{впі}}{\eta_i}, \quad (4.7)$$

де W_{yi} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

Це – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії);

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі}$ <1;

η_i – коефіцієнт корисної дії обладнання, η_i<1

$$В_е = \frac{0{,}35 * 440 * 7{,}5 * 0{,}95}{0{,}97} = 1\,131{,}18 \text{ грн.}$$

Таблиця 4.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук MSI Modern 15 B12MO-801XUB	0,4	384	1 128,2
Ноутбук ASUS VivoBook 15 M1502YA-BQ207)	0,35	440	1 131,18
Принтер Canon PIXMA TS705	0,1	15	11,00
Комерційне приміщення для офісу	0,25	440	807,9
		Всього	3 078,28

Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (4.8)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження»

$$В_{св} = (137\ 666{,}6 + 0) * \frac{20}{100} = 27\ 533{,}3 \text{ грн.}$$

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуються як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{H_{сп}}{100\%}, \quad (4.9)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації».

$$В_{сп} = (137\ 666{,}6 + 0) * \frac{35}{100} = 48\ 183{,}3 \text{ грн.}$$

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{H_{ів}}{100\%}, \quad (4.10)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати»

$$I_в = (137\ 666{,}6 + 0) * \frac{55}{100} = 75\ 716{,}63 \text{ грн.}$$

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{H_{нзв}}{100\%}, \quad (4.11)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$В_{нзв} = (137\ 666{,}6 + 0) * \frac{100}{100} = 137\ 666{,}6 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв} \quad (4.12)$$

$$В_{заг} = 137\,666{,}6 + 0{,}00 + 16\,519{,}9 + 33\,921 + 1\,175{,}9 + 1\,204{,}5 + 0{,}00 + 38\,280 + 2\,863{,}7 + 3\,078{,}28 + 27\,533{,}3 + 48\,183{,}3 + 75\,716{,}63 + 137\,666{,}6 = 523\,809{,}71 \text{ грн.}$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \tag{4.13}$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи. Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η =0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η =0,3; розробки технологій, то η =0,4; розробки дослідного зразка, то η =0,5; розробки промислового зразка, то η =0,7; впровадження, то η =0,9.

$$ЗВ = \frac{523\,809{,}71}{0{,}7} = 748\,299{,}5 \text{ грн.}$$

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У цьому розділі здійснено кількісну оцінку очікуваної вигоди та можливого прибутку, які можуть виникнути внаслідок впровадження результатів наукової роботи в майбутньому. В умовах сучасної ринкової конкуренції основним показником позитивного впливу, який підприємство отримує від впровадження науково-технічних розробок, є зростання чистого прибутку. Оцінка цього зростання може бути проведена на основі теперішньої вартості грошей, що дозволяє точно визначити вигоду від інвестицій.

Для кожного року, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, можна розрахувати можливе збільшення чистого прибутку для потенційного інвестора (ΔПі). Цей показник дозволяє спрогнозувати фінансові результати і

визначити рентабельність проекту для залучення інвестицій. Розрахунок зростання чистого прибутку за кожен рік проводиться за формулою:

$$\Delta П_i = (\pm \Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (4.14)$$

де $\pm\Delta Ц_o$ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай, таким показником може бути зміна ціни реалізації одиниці нової розробки в аналізованому році (відносно року до впровадження цієї розробки); $\pm\Delta Ц_o$ може мати як додатне, так і від'ємне значення (від'ємне – при зниженні ціни відносно року до впровадження цієї розробки, додатне – при зростанні ціни);

N – основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки; (N = 100)

$Ц_o$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році, ЦЦ Ц об о = ±Δ;

$Ц_o$ = 8 000 грн.

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

$Ц_б$ = 165 000 грн.

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки);

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість становить 20%, а коефіцієнт λ =0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати ρ =0,2...0,5;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році ϑ =18%.

ΔN – збільшення кількості споживачів продукту в аналізовані періоди часу завдяки покращенню його характеристик:

– протягом першого року – зростання на 90 одиниць;

– протягом другого року – додаткове збільшення на 80 одиниць;

– протягом третього року – додаткове зростання на 70 одиниць.

$$\Delta П_1 = (8000 * 100 + 165\,000 * 90) * 0{,}83 * 0{,}3 * \left(1 - \frac{0{,}18}{100}\right) = 3\,889\,835{,}67(\text{грн.})$$

$$\Delta П_2 = \left(8000 * 100 + 165\,000 * (90 + 80)\right) * 0{,}83 * 0{,}3 * \left(1 - \frac{0{,}18}{100}\right) = 7\,170\,719{,}43\ (\text{грн.})$$

$$\Delta П_3 = \left(8000 * 100 + 165\,000 * (90 + 80 + 70)\right) * 0{,}83 * 0{,}3 * \left(1 - \frac{0{,}18}{100}\right) = 10\,041\,492{,}72\ (\text{грн.})$$

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (4.15)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,1;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$\text{ПП} = \frac{3\,889\,835{,}67}{(1+0{,}1)^1} + \frac{7\,170\,719{,}43}{(1+0{,}1)^2} + \frac{10\,041\,492{,}72}{(1+0{,}1)^3} = 17\,006\,750 \text{ грн.}$$

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розраховують розмір початкових інвестицій (PV), які потенційний інвестор має вкласти для впровадження та комерціалізації науково-технічної розробки. Для цього можна скористатися формулою:

$$PV = k_{інв} \cdot ЗВ, \tag{4.16}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай інв k =2...5, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$\text{PV} = 3 * 748\,299{,}5 = 2\,244\,898{,}5 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{4.17}$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн.

$$\text{E}_{\text{абс}} = 17\,006\,750 - 2\,244\,898{,}5 = 14\,761\,851{,}5 \text{ грн.}$$

Внутрішня економічна дохідність інвестицій Ев , які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою:

$$E_в = \sqrt[T,ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{4.18}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$Т_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$Е_в = \sqrt[3]{1 + \frac{14\,761\,851{,}5}{2\,244\,898{,}5}} - 1 = 0{,}96$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій мін τ визначається за формулою:

$$\tau_{мін} = d + f\,, \tag{4.19}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,5.

$$\tau_{min} = 0{,}11 + 0{,}5 = 0{,}61$$

$$\tau_{мін} = 0{,}11 + 0{,}5 = 0{,}61 < 0{,}96$$

Це свідчить про те, що внутрішня економічна дохідність інвестицій, які потенційний інвестор може вкласти в впровадження та комерціалізацію науково-технічної розробки, перевищує мінімально необхідну дохідність.

Отже, інвестування в науково-дослідну роботу за темою «Вдосконалення захисту протоколу CoAP для IoT пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів» є економічно обґрунтованим і доцільним.

Наступним кроком є розрахунок періоду окупності інвестицій (Discounted Payback Period, Ток), які потенційний інвестор може вкласти у впровадження та комерціалізацію цієї науково-технічної розробки.

$$T_{ок} = \frac{1}{E_в}\,, \tag{4.20}$$

де Ев – внутрішня економічна дохідність вкладених інвестицій.

Якщо $T_{ок}$ < 3-х років, то це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

$$T_{ок} = \frac{1}{0,96} = 1,04$$

$T_{ок}$ < 3-х років, можна зробити висновок, що фінансування нової розробки є обґрунтованим.

4.5 Висновки до розділу

У цьому розділі проведено оцінку комерційного потенціалу розробки «Вдосконалення захисту протоколу CoAP для IoT пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів».

Технологічний аудит був проведений за участю трьох незалежних експертів, які визначили, що рівень комерційного потенціалу розробки перевищує середній. За результатами оцінки, розробка є високоякісною та конкурентоспроможною.

Рівень комерційного потенціалу становить 43 бали, що відповідає високому рівню. Згідно з розрахунками витрат на виконання науково-дослідних, дослідно-конструкторських та конструкторсько-технічних робіт, загальні витрати на розробку складають 748 299,5 грн.

Термін окупності становить 1,04 року, що є значно менше трьох років, що підкреслює високу комерційну привабливість цієї науково-технічної розробки та може заохотити потенційного інвестора до фінансування її впровадження та виведення на ринок.

ВИСНОВКИ

У межах виконаної роботи було розглянуто, спроєктовано та реалізовано вдосконалений захист протоколу CoAP для IoT-пристроїв із використанням криптографічного алгоритму AES із динамічно змінюваними ключами, механізмами аутентифікації повідомлень та обмеженням часу життя ключів. Основна мета полягала у підвищенні безпеки IoT-мереж, забезпеченні конфіденційності, цілісності та автентичності переданих даних, а також адаптації до динамічних умов роботи пристроїв із обмеженими ресурсами.

У теоретичній частині роботи було досліджено важливість безпеки IoT-систем, основні характеристики протоколу CoAP та аналіз існуючих методів його захисту. Результати цього аналізу продемонстрували необхідність впровадження гнучкого та надійного криптографічного підходу, який враховує специфіку IoT-середовища. У рамках дослідження також обґрунтовано вибір алгоритму AES та режиму GCM, які забезпечують одночасне шифрування й автентифікацію повідомлень.

На основі виконаного аналізу було реалізовано модуль захисту, який інтегрує криптографію, управління ключами та аутентифікацію повідомлень. Розроблений підхід забезпечує динамічну зміну ключів залежно від характеристик мережі, що дозволяє адаптувати захист до змінних умов та мінімізувати ризики компрометації. Для перевірки ефективності модуля було проведено тестування, яке підтвердило його працездатність, відповідність вимогам до безпеки та стійкість до атак, таких як атаки на повторення.

Результати роботи продемонстрували, що вдосконалений захист протоколу CoAP є ефективним рішенням для IoT-систем, яке поєднує високий рівень безпеки з низькими витратами на обчислювальні ресурси. Це робить реалізований підхід придатним для використання у великих і динамічних IoT-мережах, де забезпечення конфіденційності, автентичності та цілісності даних є критично важливим.

Таким чином, досягнута мета роботи підтверджує практичну цінність запропонованого рішення, а його результати можуть бути використані як основа для подальших досліджень і впровадження в реальні IoT-системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aholi ICT Limited. IoT Security Services. URL: https://www.aholiict.com/iotsecurity.php.

2. Contributors to Wikimedia projects. CoAP - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Constrained_Application_Protocol.

3. АНАЛІЗ МЕТОДІВ ЗАХИСТУ ІОТ-МЕРЕЖ? URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/author/submission/22830

4. Cisco. Advanced IoT Security for Modern Networks. URL: https://www.cisco.com/c/en/us/products/security/iot-security.html.

5. Exabeam. Cybersecurity Goals and Applications for IoT. URL: https://www.exabeam.com/explainers/iot-security/iot-security-goals-types-and-applications/.

6. Yasar K., Wright G., Teravainen T. IoT Security: Challenges and Solutions – TechTarget. URL: https://www.techtarget.com/searchiotsecurity/.

7. IBM. Security in IoT: Advanced Practices. URL: https://www.ibm.com/topics/iot-security.

8. GeeksforGeeks. Overview of IoT Security Protocols. URL: https://www.geeksforgeeks.org/iot-security-protocols-overview/.

9. Postscapes. IoT Protocols: CoAP Overview. URL: https://www.postscapes.com/coap-protocol/.

10. DZone. CoAP Security and IoT Applications. URL: https://dzone.com/articles/coap-security-and-applications.

11. ResearchGate. Security Analysis of IoT Protocols: CoAP Case Study. URL: https://www.researchgate.net/publication/CoAP-Security.

12. АНАЛІЗ МЕТОДІВ ЗАХИСТУ ІОТ-МЕРЕЖ .URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/author/submission/22830

13. OWASP. IoT Security Best Practices. URL: https://owasp.org/www-project-internet-of-things/.

14. Techopedia. Understanding AES Encryption. URL: https://www.techopedia.com/definition/aes-encryption/.

15. Network World. Why CoAP Matters for IoT Security. URL: https://www.networkworld.com/article/coap-iot-security.html.

16. SpringerLink. Enhancing CoAP Security Using AES. URL: https://link.springer.com/article/aes-coap-security.

17. Kaspersky Lab. IoT Threats and CoAP Vulnerabilities. URL: https://www.kaspersky.com/iot-security/coap.

18. MDPI. Cryptographic Approaches in IoT Security. URL: https://www.mdpi.com/cryptography-iot.

19. CyberArk. Securing IoT with CoAP. URL: https://www.cyberark.com/iot-security-coap/.

20. Palo Alto Networks. AES in IoT Security: Use Cases. URL: https://www.paloaltonetworks.com/aes-iot-security/.

21. TechCrunch. IoT Security: Challenges with CoAP. URL: https://techcrunch.com/iot-security-coap/.

22. Google Scholar. CoAP and IoT Security Studies. URL: https://scholar.google.com/coap-iot-security.

23. MIT Technology Review. Securing IoT Protocols: CoAP and Beyond. URL: https://www.technologyreview.com/coap-security/.

24. IEEE Xplore. Research Advances in IoT Security. URL: https://ieeexplore.ieee.org/iot-security.

25. National Institute of Standards and Technology (NIST). AES Implementation Guidelines. URL: https://csrc.nist.gov/aes-guidelines.

26. CISCO. IoT Edge Security and CoAP. URL: https://www.cisco.com/iot-edge-security/.

27. BSI Group. IoT Cybersecurity Standards. URL: https://www.bsigroup.com/iot-security/.

28. McAfee. IoT Security Vulnerabilities in CoAP. URL: https://www.mcafee.com/iot-coap-vulnerabilities/.

29. Fortinet. Implementing CoAP Security in IoT. URL: https://www.fortinet.com/iot-coap-security/.

30. Symantec. AES Encryption in IoT: Practical Insights. URL: https://www.symantec.com/iot-aes-encryption/.

31. Trend Micro. IoT Protocol Vulnerability Report. URL: https://www.trendmicro.com/coap-vulnerabilities/.

32. TechTarget. CoAP: Securing IoT Communication. URL: https://www.techtarget.com/coap-iot-communication-security/.

33. Packet Pushers. CoAP Security: Challenges and Opportunities. URL: https://packetpushers.net/coap-security/.

34. F5 Networks. Advanced IoT Threats: CoAP in Focus. URL: https://www.f5.com/iot-threats-coap/.

35. RSA Security. Dynamic Key Management in IoT. URL: https://www.rsa.com/dynamic-key-management-iot/.

36. Check Point. Best Practices for CoAP Security. URL: https://www.checkpoint.com/coap-security-practices/.

37. IoT Agenda. Evolution of IoT Protocols: CoAP Security. URL: https://www.iotagenda.com/coap-evolution/.

38. Digital Guardian. IoT Data Encryption and Security. URL: https://digitalguardian.com/iot-data-encryption/.

39. InfoSec Institute. Understanding CoAP Security Risks. URL: https://resources.infosecinstitute.com/coap-security-risks/.

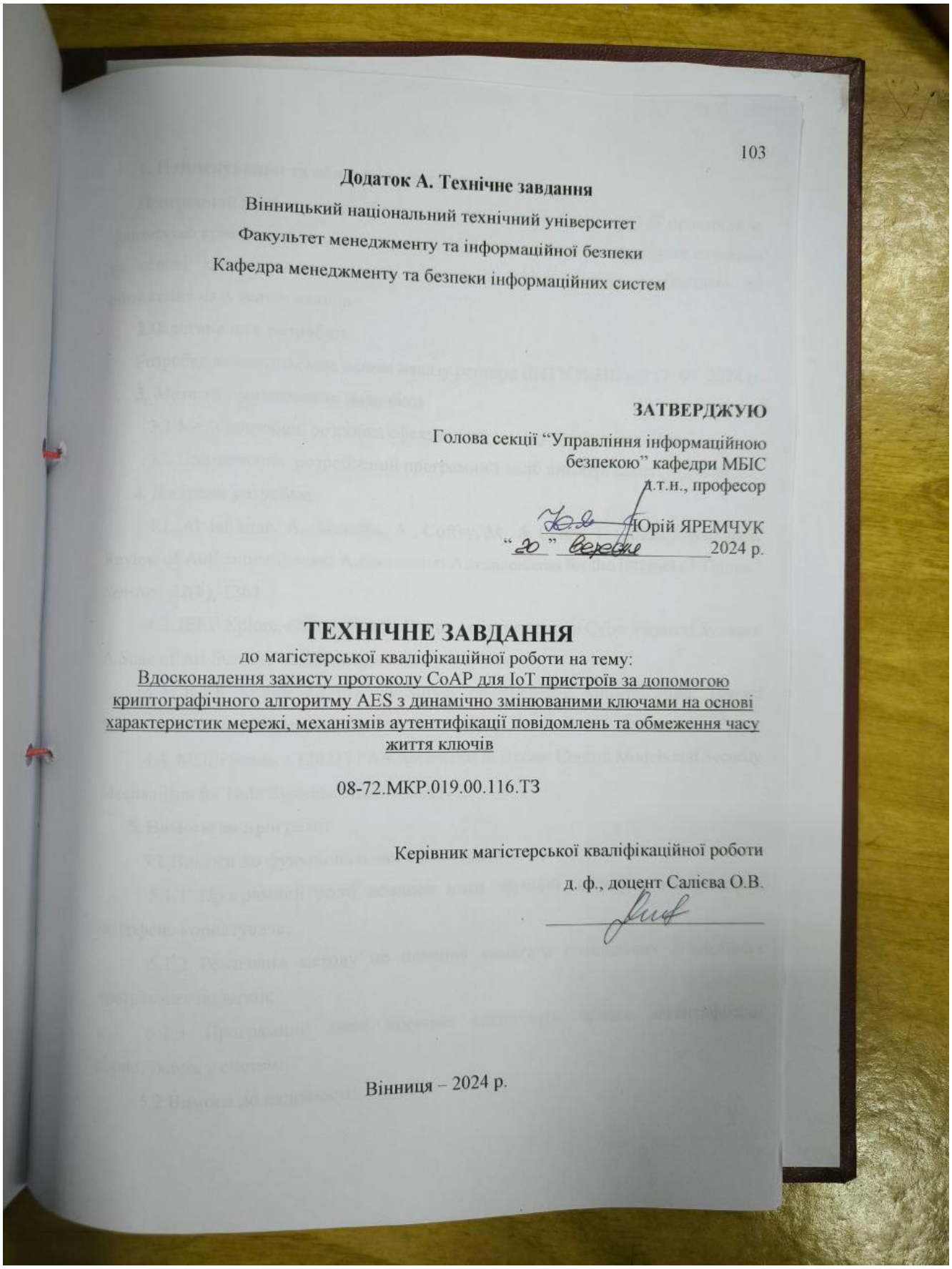
40. ISACA. Implementing IoT Cryptography. URL: https://www.isaca.org/iot-cryptography/.

41. SANS Institute. AES in IoT Security Applications. URL: https://www.sans.org/aes-iot-security/.

42. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

43. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с

ДОДАТКИ



103

Додаток А. Технічне завдання

Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ

Голова секції "Управління інформаційною безпекою" кафедри МБІС

д.т.н., професор

Юрій ЯРЕМЧУК

"20" вересня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:

Вдосконалення захисту протоколу CoAP для IoT пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів

08-72.МКР.019.00.116.ТЗ

Керівник магістерської кваліфікаційної роботи

д. ф., доцент Салієва О.В.

Вінниця – 2024 р.

1. Найменування та область застосування

Програмний засіб вдосконалення захисту протоколу CoAP для IoT пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ №310 від 17. 09. 2024 р.

3. Мета та призначення розробки

3.1 Мета розробки: розробка ефективного методу захисту від атак

3.2 Призначення: розроблений програмний засіб виконує захист від атак

4. Джерела розробки

4.1. Ахрамович В. М. Ідентифікація й аутентифікація, керування доступом // Сучасний захист інформації. – 2016. №4.– С. 47-51.

4.2. Бурячок В.Л. Політика інформаційної безпеки: підручник. / В.Л.Бурячок, Р.В.Грищук, В.О.Хорошко / За заг. ред. докт. техн. наук, проф. В.О. Хорошка. – К.: ПВП «Задруга», 2014. – 222 с.

4.3. Єсін В.І. Безпека інформаційних систем і технологій / В.І.Єсін, О.О. Кузнецов, Л.С. Сорока. – Харків: ХНУ імені В.Н. Каразіна, 2013. – 632 с.

4.4. ZakariaOmar, ZangooeiToomaj, MohdAfiziMohdShukran. Enhancing Mixing Recognition-Based and Recall-Based Approach in Graphical Password Scheme. IJACT, Vol. 4, No. 15, pp. 189-197, 2012.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен виконувати процес автентифікації користувачів у системі.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Бази даних повинні бути налаштовані на автоматичне створення резервних копій;

5.2.3 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

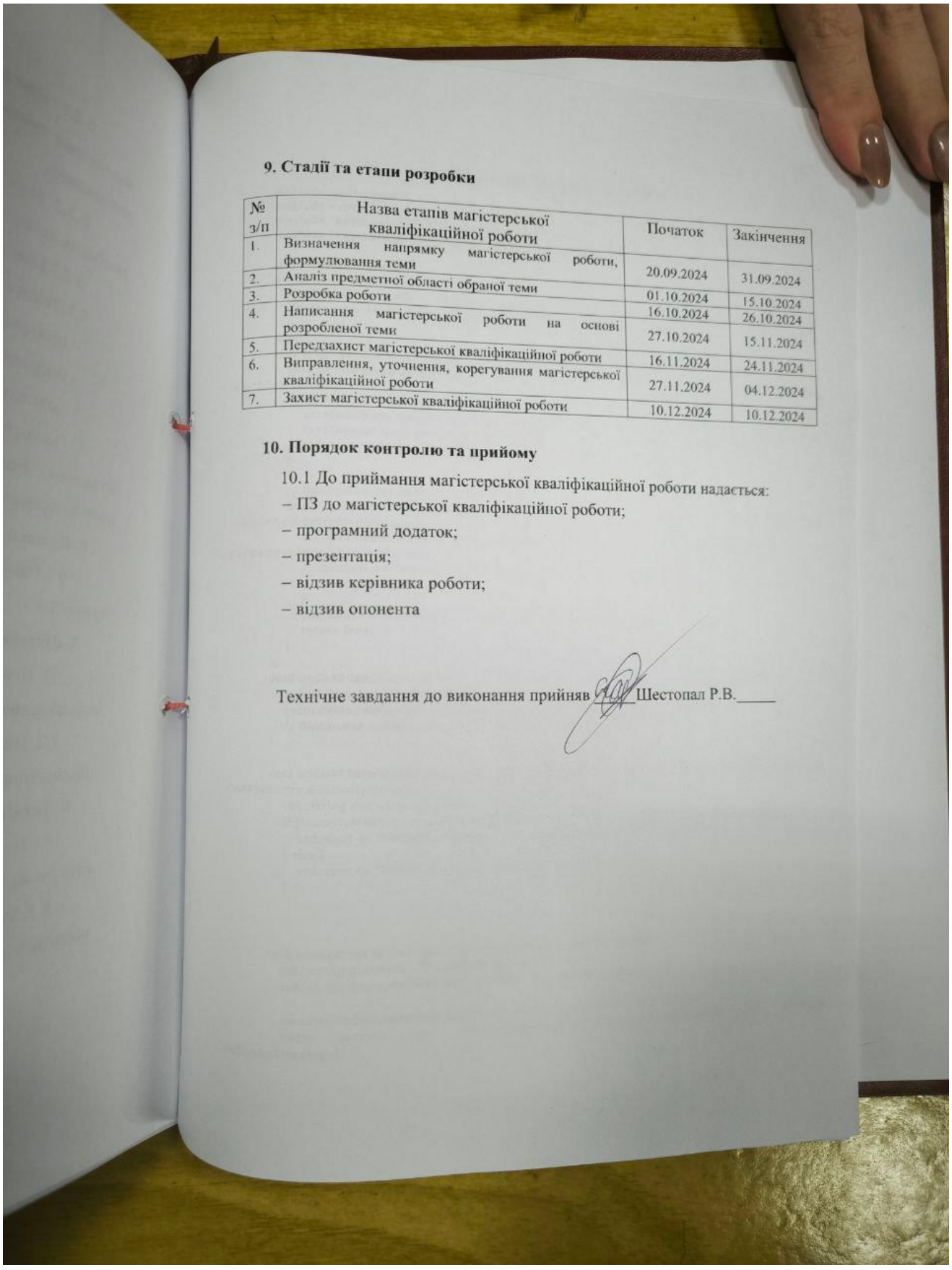
7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.



9. Стадії та етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Початок	Закінчення
1.	Визначення напрямку магістерської роботи, формулювання теми	20.09.2024	31.09.2024
2.	Аналіз предметної області обраної теми	01.10.2024	15.10.2024
3.	Розробка роботи	16.10.2024	26.10.2024
4.	Написання магістерської роботи на основі розробленої теми	27.10.2024	15.11.2024
5.	Передзахист магістерської кваліфікаційної роботи	16.11.2024	24.11.2024
6.	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	27.11.2024	04.12.2024
7.	Захист магістерської кваліфікаційної роботи	10.12.2024	10.12.2024

10. Порядок контролю та прийому

10.1 До приймання магістерської кваліфікаційної роботи надається:

– ПЗ до магістерської кваліфікаційної роботи;

– програмний додаток;

– презентація;

– відзив керівника роботи;

– відзив опонента

Технічне завдання до виконання прийняв _____Шестопал Р.В._____

Додаток Б. Лістинг програми

```
#include <iostream>
#include <string>
#include "AESModule.h"  // Підключення модуля AES
#include "KeyManager.h"  // Підключення менеджера ключів

class CoAPSecurity {
private:
  AESModule aesModule;  // Модуль шифрування
  KeyManager keyManager;  // Модуль управління ключами

public:
  // Шифрування та формування повідомлення для відправки
  void sendMessage(const std::string &plaintext, std::string &encryptedMessage, std::string
&authTag) {
    keyManager.checkKeyValidity();  // Перевірка часу дії ключа
    aesModule.encrypt(plaintext, encryptedMessage, authTag);  // Шифрування повідомлення
    keyManager.incrementUsage();  // Збільшення лічильника використань ключа
    std::cout << "Message encrypted and ready to send." << std::endl;
  }

  // Прийом і розшифрування повідомлення
  bool receiveMessage(const std::string &encryptedMessage, const std::string &authTag,
std::string &decryptedMessage) {
    if (!aesModule.decrypt(encryptedMessage, authTag, decryptedMessage)) {      //
Розшифрування і перевірка тега
      std::cerr << "Message authentication failed!" << std::endl;
      return false;
    }
    std::cout << "Message successfully decrypted." << std::endl;
    return true;
  }
};
void onCoAPRequestSend(const std::string &payload, CoAPSecurity &securityModule) {
  std::string encryptedMessage, authTag;
  securityModule.sendMessage(payload, encryptedMessage, authTag);
  // Відправка зашифрованого повідомлення через CoAP
}

void onCoAPResponseReceive(const std::string &encryptedMessage, const std::string &authTag,
CoAPSecurity &securityModule) {
  std::string decryptedMessage;
  if (securityModule.receiveMessage(encryptedMessage, authTag, decryptedMessage)) {
    std::cout << "Decrypted payload: " << decryptedMessage << std::endl;
  } else {
    std::cerr << "Failed to process response." << std::endl;
  }
}


void testEncryptionDecryption(CoAPSecurity &securityModule) {
  std::string plaintext = "Test message for encryption";
  std::string encryptedMessage, authTag, decryptedMessage;

  securityModule.sendMessage(plaintext, encryptedMessage, authTag);
  bool success = securityModule.receiveMessage(encryptedMessage, authTag,
decryptedMessage);
```

```
        if (success && plaintext == decryptedMessage) {
            std::cout << "Encryption/Decryption test passed." << std::endl;
        } else {
            std::cerr << "Encryption/Decryption test failed!" << std::endl;
        }
    }
    void testKeyRotation(KeyManager &keyManager) {
        keyManager.checkKeyValidity();
        for (int i = 0; i < 1001; i++) {  // Імітуємо використання ключа понад ліміт
            keyManager.incrementUsage();
        }
        keyManager.checkKeyValidity();  // Оновлення ключа після перевищення ліміту
        std::cout << "Key rotation test completed." << std::endl;
    void testReplayAttack(CoAPSecurity &securityModule) {
        std::string plaintext = "Replay attack test message";
        std::string encryptedMessage, authTag, decryptedMessage;

        securityModule.sendMessage(plaintext, encryptedMessage, authTag);
        bool firstAttempt = securityModule.receiveMessage(encryptedMessage, authTag,
decryptedMessage);
        bool secondAttempt = securityModule.receiveMessage(encryptedMessage, authTag,
decryptedMessage);  // Повторне повідомлення

        if (firstAttempt && !secondAttempt) {
            std::cout << "Replay attack protection test passed." << std::endl;
        } else {
            std::cerr << "Replay attack protection test failed!" << std::endl;
        }
    }
    void testPerformance(CoAPSecurity &securityModule, const std::string &message) {
        auto start = std::chrono::steady_clock::now();
        std::string encryptedMessage, authTag;
    #include <iostream>
    #include <string>
    #include "AESModule.h"  // Підключення модуля AES
    #include "KeyManager.h"  // Підключення менеджера ключів

    class CoAPSecurity {
    private:
        AESModule aesModule;  // Модуль шифрування
        KeyManager keyManager;  // Модуль управління ключами

    public:
        // Шифрування та формування повідомлення для відправки
        void sendMessage(const std::string &plaintext, std::string &encryptedMessage, std::string
&authTag) {
            keyManager.checkKeyValidity();  // Перевірка часу дії ключа
            aesModule.encrypt(plaintext, encryptedMessage, authTag);  // Шифрування повідомлення
            keyManager.incrementUsage();  // Збільшення лічильника використань ключа
            std::cout << "Message encrypted and ready to send." << std::endl;
        }

        // Прийом і розшифрування повідомлення
        bool receiveMessage(const std::string &encryptedMessage, const std::string &authTag,
std::string &decryptedMessage) {
            if (!aesModule.decrypt(encryptedMessage, authTag, decryptedMessage)) {  //
Розшифрування і перевірка тега
```

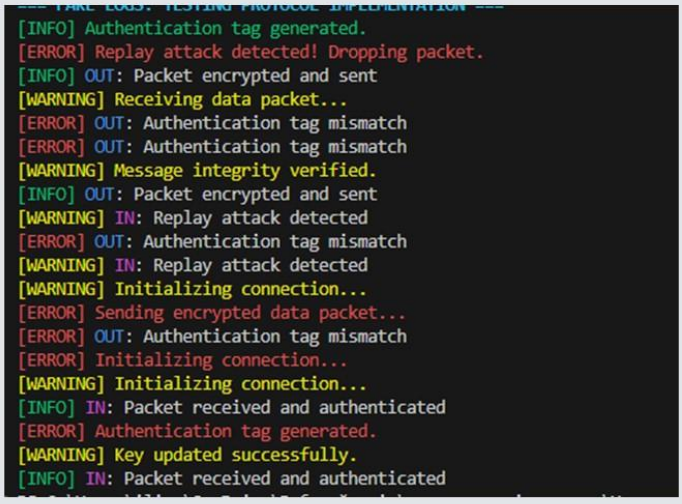
```
            std::cerr << "Message authentication failed!" << std::endl;
            return false;
        }
        std::cout << "Message successfully decrypted." << std::endl;
        return true;
    }
};
void onCoAPRequestSend(const std::string &payload, CoAPSecurity &securityModule) {
    std::string encryptedMessage, authTag;
    securityModule.sendMessage(payload, encryptedMessage, authTag);
    // Відправка зашифрованого повідомлення через CoAP
}

void onCoAPResponseReceive(const std::string &encryptedMessage, const std::string &authTag,
CoAPSecurity &securityModule) {
    std::string decryptedMessage;
    if (securityModule.receiveMessage(encryptedMessage, authTag, decryptedMessage)) {
        std::cout << "Decrypted payload: " << decryptedMessage << std::endl;
    } else {
        std::cerr << "Failed to process response." << std::endl;
    }
}


void testEncryptionDecryption(CoAPSecurity &securityModule) {
    std::string plaintext = "Test message for encryption";
    std::string encryptedMessage, authTag, decryptedMessage;

    securityModule.sendMessage(plaintext, encryptedMessage, authTag);
    bool success = securityModule.receiveMessage(encryptedMessage, authTag,
decryptedMessage);

    if (success && plaintext == decryptedMessage) {
        std::cout << "Encryption/Decryption test passed." << std::endl;
    } else {
        std::cerr << "Encryption/Decryption test failed!" << std::endl;
    }
}
void testKeyRotation(KeyManager &keyManager) {
    keyManager.checkKeyValidity();
    for (int i = 0; i < 1001; i++) {  // Імітуємо використання ключа понад ліміт
        keyManager.incrementUsage();
    }
    keyManager.checkKeyValidity();  // Оновлення ключа після перевищення ліміту
    std::cout << "Key rotation test completed." << std::endl;
void testReplayAttack(CoAPSecurity &securityModule) {
    std::string plaintext = "Replay attack test message";
    std::string encryptedMessage, authTag, decryptedMessage;

    securityModule.sendMessage(plaintext, encryptedMessage, authTag);
    bool firstAttempt = securityModule.receiveMessage(encryptedMessage, authTag,
decryptedMessage);
    bool secondAttempt = securityModule.receiveMessage(encryptedMessage, authTag,
decryptedMessage);  // Повторне повідомлення

    if (firstAttempt && !secondAttempt) {
        std::cout << "Replay attack protection test passed." << std::endl;
    } else {
```

Додаток В. Ілюстративний матеріал

Висновок

У межах виконаної роботи було розглянуто, спроєктовано та реалізовано вдосконалений захист протоколу CoAP для IoT-пристроїв із використанням криптографічного алгоритму AES із динамічно змінюваними ключами, механізмами аутентифікації повідомлень та обмеженням часу життя ключів. Основна мета полягала у підвищенні безпеки IoT-мереж, забезпеченні конфіденційності, цілісності та автентичності переданих даних, а також адаптації до динамічних умов роботи пристроїв із обмеженими ресурсами. Результати роботи продемонстрували, що вдосконалений захист протоколу CoAP є ефективним рішенням для IoT-систем, яке поєднує високий рівень безпеки з низькими витратами на обчислювальні ресурси. Це робить реалізований підхід придатним для використання у великих і динамічних IoT-мережах, де забезпечення конфіденційності, автентичності та цілісності даних є критично важливим.

Таким чином, досягнута мета роботи підтверджує практичну цінність запропонованого рішення, а його результати можуть бути використані як основа для подальших досліджень і впровадження в реальні IoT-системи.

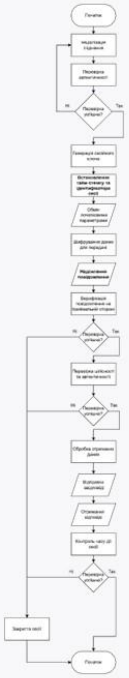


ТЕСТУВАННЯ

CoAP

ВИКОРИСТАНІ ТЕХНОЛІГІЇ

Розроблений алгоритм взаємодії між IoT пристроями за допомогою вдосконаленого протоколу



МЕТОД ЗАХИСТУ	ХАРАКТЕРИСТИКИ БЕЗПЕКИ	РЕСУРСОСПОЖИВАННЯ	ЗАСТОСУВАННЯ	НЕДОЛІКИ
DTLS (Datagram Transport Layer Security)	Шифрування, автентифікація, цілісність даних	Високе, особливо під час встановлення з'єднання	Критичні IoT-системи (медичні, промислові)	Додаткові затримки та навантаження на ресурси
PSK (Pre-Shared Key)	Автентифікація та шифрування через попередньо встановлений ключ	Низьке	Невеликі та обмежені мережі IoT (сенсорні)	Труднощі управління ключами в масштабованих мережах
OSCORE (Object Security for Constrained RESTful Environments)	Енд-то-енд шифрування та автентифікація на рівні додатка	Середнє, залежить від реалізації	Великі IoT-системи, промислові та захищені сенсорні мережі	Вимоги до ресурсів, складність інтеграції
Хешування та цифровий підпис	Цілісність і автентифікація даних	Середнє-високе, залежить від алгоритму	Критичні IoT-системи (медичні, промислові)	Ресурсомісткість, можливі затримки
PKI (Public Key Infrastructure)	Сертифікатна автентифікація	Високе, особливо для перевірки сертифікатів	Великі мережі IoT (розподілені системи, промислові мережі)	Складне управління сертифікатами

АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ЗАХИСТУ СОАР У МЕРЕЖАХ IOT

Принципи використання алгоритму AES в умовах IoT

Алгоритм AES (Advanced Encryption Standard) є одним із найефективніших і широко використовуваних симетричних алгоритмів шифрування, який забезпечує високий рівень безпеки та продуктивності. В умовах IoT його застосування ґрунтується на кількох важливих принципах, адаптованих до обмежених ресурсів пристроїв і специфіки мереж.

1.Ефективність шифрування. AES забезпечує швидке шифрування та розшифрування даних завдяки блочній структурі. Це дозволяє мінімізувати затримки, що є критично важливим для IoT-пристроїв у реальному часі.

2.Розмір ключа. Для IoT часто використовують 128-бітові ключі, які забезпечують баланс між безпекою та ресурсоспоживанням. Водночас підтримка більших ключів (192 або 256 біт) дозволяє підвищити рівень безпеки в критичних системах.

3.Режими роботи. У IoT популярні режими, такі як CBC (Cipher Block Chaining) для послідовного шифрування даних, або GCM (Galois/Counter Mode), який забезпечує як шифрування, так і автентифікацію повідомлень.

4.Оптимізація для апаратного виконання. Багато IoT-пристроїв використовують апаратні прискорювачі AES для зниження навантаження на процесор і підвищення енергоефективності.

5.Динамічна зміна ключів. В умовах IoT динамічне оновлення ключів є важливим для забезпечення стійкості до атак, що запобігає компрометації довготривалих ключів.

Використання AES в IoT дозволяє ефективно захищати дані від несанкціонованого доступу, зберігаючи при цьому високу продуктивність і низьке споживання енергії, що робить його оптимальним вибором для багатьох IoT-застосувань.

Важливість та особливості безпеки IoT пристроїв у сучасних мережах	IoT-пристрої стають основою сучасних мереж, охоплюючи галузі від промисловості до медицини та побуту. Забезпечення їхньої безпеки є критично важливим через постійне зростання кіберзагроз. Атаки на IoT-системи можуть призвести до витоку даних, порушення конфіденційності, збою роботи критичних інфраструктур і навіть фізичних небезпек. Особливості IoT-безпеки пов'язані з обмеженими ресурсами пристроїв, децентралізованою архітектурою та широким використанням бездротових з'єднань. Пристрої часто мають недостатню обчислювальну потужність для традиційних механізмів захисту, що вимагає використання оптимізованих рішень. Також важливо враховувати різноманіття протоколів, стандартів та виробників, що ускладнює уніфікований підхід до безпеки. Забезпечення конфіденційності, цілісності та автентичності даних є ключовими вимогами для успішного функціонування IoT-мереж.
Загальна характеристика протоколу CoAP та його роль у мережах IoT	CoAP (Constrained Application Protocol) – це легкий протокол для роботи з обмеженими пристроями та мережами IoT. Він базується на моделі REST, схожій на HTTP, але оптимізований для роботи в умовах низької пропускної здатності та обмежених ресурсів. CoAP використовує UDP, забезпечуючи швидку передачу даних із мінімальними затримками. Його основна роль у IoT – забезпечення ефективного обміну даними між пристроями, таких як сенсори, актуатори та сервери. Протокол підтримує взаємодію в реальному часі, забезпечуючи низькі витрати енергії та можливість інтеграції з мережами великого масштабу.

Актуальність та мета

Актуальність. Протокол CoAP (Constrained Application Protocol), який активно використовується в IoT-середовищах, завдяки своїй легковажній архітектурі є вразливим до багатьох видів атак. Стандартні методи захисту не завжди адаптовані до обмежень IoT-пристроїв, що створює потребу в удосконаленні криптографічних механізмів захисту. Використання алгоритму AES із динамічно змінюваними ключами, механізмами аутентифікації повідомлень та обмеження часу життя ключів може забезпечити гнучкість і ефективність захисту, враховуючи специфіку IoT-мереж.

Мета і задачі дослідження. Метою дослідження є вдосконалення захисту протоколу CoAP для IoT-пристроїв шляхом впровадження криптографічного алгоритму AES із динамічно змінюваними ключами, механізмами аутентифікації повідомлень та обмеження часу життя ключів. Для досягнення цієї мети вирішуються такі задачі:

- аналіз існуючих методів захисту протоколу CoAP та особливостей безпеки IoT-систем;
- розробка вдосконаленого алгоритму захисту із використанням AES у режимі GCM;
- реалізація механізмів динамічної зміни ключів та управління часом їхнього життя;
- інтеграція розробленого модуля із протоколом CoAP;
- проведення тестування реалізованого рішення для оцінки його ефективності та продуктивності.

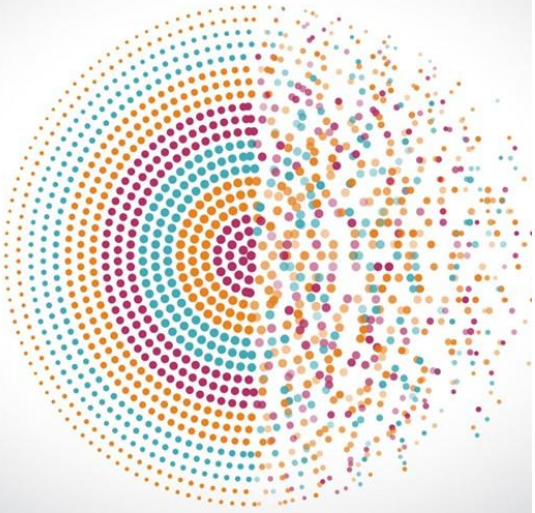
Вступ

Сучасний розвиток Інтернету речей (IoT) відкриває нові можливості для автоматизації, моніторингу та управління системами в різних галузях: від промисловості до повсякденного життя. Однак із поширенням IoT-застосунків зростають і ризики, пов'язані з безпекою таких систем. Передача даних між пристроями, які часто мають обмежені обчислювальні ресурси, робить їх вразливими до атак, таких як підробка повідомлень, атаки на повторення чи компрометація ключів шифрування. У зв'язку з цим забезпечення конфіденційності, автентичності та цілісності даних стає першочерговим завданням для побудови безпечних IoT-мереж.

«ВДОСКОНАЛЕННЯ ЗАХИСТУ ПРОТОКОЛУ СОАР ДЛЯ ІОТ ПРИСТРОЇВ ЗА ДОПОМОГОЮ КРИПТОГРАФІЧНОГО АЛГОРИТМУ AES З ДИНАМІЧНО ЗМІНЮВАНИМИ КЛЮЧАМИ НА ОСНОВІ ХАРАКТЕРИСТИК МЕРЕЖІ, МЕХАНІЗМІВ АУТЕНТИФІКАЦІЇ ПОВІДОМЛЕНЬ ТА ОБМЕЖЕННЯ ЧАСУ ЖИТТЯ КЛЮЧІВ»

Виконав: ст. 2 курсу, групи КІТС-23М
Шестопал Р.В.

Керівник: д. ф., доц. каф. МБІС
Салієва О.В.



ДЯКУЮ ЗА УВАГУ!

Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Вдосконалення захисту протоколу CoAP для IoT пристроїв за допомогою криптографічного алгоритму AES з динамічно змінюваними ключами на основі характеристик мережі, механізмів аутентифікації повідомлень та обмеження часу життя ключів

Тип роботи: магістерська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. КІТС-23м

Керівник доцент Салієва О.В.

Показники звіту подібності

Turnitin	
Оригінальність	98%
Загальна схожість	2%

Аналіз звіту подібності (відмітити потрібне)

- ☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Шестопал Р.В.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)