Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол №__ від __ грудня 2025р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)

«___» _____ 2025 р.

Кваліфікаційна робота

здобувача другого (магістерського) рівня вищої освіти

«ОЦІНКА ТА ПІДВИЩЕННЯ СТІЙКОСТІ ARDUINO-БАЗОВАНИХ IOT-СИСТЕМ ДО БАЗОВИХ МЕРЕЖЕВИХ АТАК»

Спеціальність ***174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка***
Освітня програма ***Комп'ютеризовані системи управління та автоматика***

Виконавець _____ **Р. В. ШУЛЬГА**
(підпис)

Науковий керівник _____ **М. М. ХРУСЛОВ**
(підпис)

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи магістра складається зі вступу, трьох розділів, висновків, списку використаних джерел і трьох додатків. Загальний обсяг роботи становить 80 сторінки, із яких 70 сторінок основної частини. Робота містить 17 рисунків, 2 таблиці та список використаних джерел із 31 найменувань.

Метою роботи є підвищення рівня захищеності та стійкості Arduino-базованих IoT-систем в умовах впливу мережевих атак шляхом розроблення комплексної методики оцінювання та впровадження ефективних механізмів захисту.

Об'єктом дослідження є процес функціонування та інформаційного обміну в локальних IoT-мережах на базі мікроконтролерів.

Предметом дослідження є методи виявлення та протидії мережевим загрозам, моделі атак на протоколи IoT (ARP-spoofing, DoS, MQTT-вразливості), а також метрики оцінювання стійкості системи (доступність, цілісність, затримки передачі даних).

Наукова та практична задача, що вирішується у роботі, полягає у створенні відтворюваного підходу до тестування та захисту IoT-пристроїв, який дозволяє мінімізувати ресурсомісткість налаштувань, зменшити час простою під час атак та забезпечити доказову ефективність впроваджених контрзаходів.

Область застосування результатів включає прототипування та експлуатацію IoT-рішень у навчальних лабораторіях, малому та середньому бізнесі.

Ключові слова: Arduino, ESP8266/ESP32, IoT, MQTT, TLS, HMAC, IDS, Suricata, Snort, Mosquitto, ARP-spoofing, DoS, DNS-спуфінг, rate limiting, ACL, стійкість, мережеві атаки.

ABSTRACT

The explanatory note of the Master's thesis consists of an introduction, three chapters, conclusions, a list of references, and three appendices. The total volume of the work is 80 pages, of which 70 pages constitute the main body. The thesis contains 17 figures, 2 tables, and a list of references comprising 31 items.

The **purpose** of the work is to enhance the security and resilience of Arduino-based IoT systems under the influence of network attacks by developing a comprehensive evaluation methodology and implementing effective protection mechanisms.

The **object of research** is the process of functioning and information exchange in local IoT networks based on microcontrollers.

The **subject of research** covers methods for detecting and neutralizing network threats, attack models on IoT protocols (ARP-spoofing, DoS, MQTT vulnerabilities), as well as metrics for assessing system resilience (availability, integrity, data transmission latency).

The **scientific and practical problem** addressed in the thesis lies in creating a reproducible approach to testing and protecting IoT devices, which allows minimizing configuration resource requirements, reducing downtime during attacks, and ensuring the proven effectiveness of implemented countermeasures.

The **field of application** of the results includes prototyping and operation of IoT solutions in educational laboratories, small and medium-sized businesses.

Keywords: Arduino, ESP8266/ESP32, IoT, MQTT, TLS, HMAC, IDS, Suricata, Snort, Mosquitto, ARP-spoofing, DoS, DNS-spoofing, rate limiting, ACL, resilience, network attacks.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ACL	–	Access Control List — список контролю доступу (правила доступу на брокері/маршрутизаторі).
ARP	–	Address Resolution Protocol — протокол зіставлення IP ↔ MAC; основа ARP-spoofing.
CA	–	Certificate Authority — центр сертифікації X.509.
CIA	–	Confidentiality, Integrity, Availability — тріада безпеки: конфіденційність, цілісність, доступність.
CoAP	–	Constrained Application Protocol — протокол для обмежених пристроїв (UDP, REST-подібний).
DNS	–	Domain Name System — система доменних імен; ціль для DNS-spoofing.
DoS/DDoS	–	(Distributed) Denial of Service — (розподілена) відмова в обслуговуванні.
ESP8266/ESP32	–	Wi-Fi-SoC від Espressif для IoT-вузлів.
F1	–	F1-score — гармонійне середнє точності (Precision) та повноти (Recall).
FPR	–	False Positive Rate — частка хибних спрацьовувань системи виявлення.
HMAC	–	Hash-based Message Authentication Code — код автентифікації повідомлень на хеш-функції.
HTTP/HTTPS	–	HyperText Transfer Protocol (Secure) — прикладний протокол (з TLS для HTTPS).
ICMP	–	Internet Control Message Protocol — службові повідомлення (пінг тощо).
IDS/IPS	–	Intrusion Detection/Prevention System — система виявлення/запобігання вторгнень (напр., Suricata, Snort).
IoT	–	Internet of Things — Інтернет речей.
IP	–	Internet Protocol — мережевий протокол (IPv4/IPv6).

iptables/nftables	–	Підсистеми фільтрації/маршрутизації пакетів у Linux.
MAC (адреса)	–	Media Access Control address — апаратна адреса мережевого інтерфейсу.
MITM	–	Man-In-The-Middle — атака «людина посередині».
Mosquitto	–	Популярний брокер MQTT (ідентифікація, ACL, TLS, плагіни).
MTTR	–	Mean Time To Recovery — середній час відновлення працездатності.
PSK	–	Pre-Shared Key — попередньо узгоджений ключ (TLS-PSK, WPA2-PSK).
QoS (MQTT)	–	Quality of Service — рівень доставки повідомлень 0/1/2.
ROC	–	Receiver Operating Characteristic — характеристика робочих точок класифікатора.
RTT	–	Round-Trip Time — час оберненого проходження пакета.
Snort	–	Сигнатурна IDS/IPS (правила, журнали тривог).
SYN-flood	–	Тип DoS-атаки, що зловживає TCP-рукостисканням (SYN).
TCP	–	Transmission Control Protocol — надійний транспортний протокол.
TLS	–	Transport Layer Security — протокол шифрування/автентифікації (сертифікати/X.509, PSK).
TPR	–	True Positive Rate — частка істинно виявлених атак (Recall).
UDP	–	User Datagram Protocol — ненадійний транспортний протокол (мінімальні накладні витрати).
UART	–	Universal Asynchronous Receiver/Transmitter — асинхронний послідовний інтерфейс.
VLAN	–	Virtual LAN — віртуальне сегментування мережі.
VPN	–	Virtual Private Network — захищений тунель зв’язку.

WPA2/WPA3	–	Wi-Fi Protected Access — стандарти захисту бездротових мереж.
Wireshark	–	Аналізатор мережевого трафіку (PCAP, фільтри).
WDT	–	Watchdog Timer — сторожовий таймер від зависань.
X.509	–	Стандарт сертифікатів відкритих ключів.
Bettercap	–	Інструмент MITM/спуфінгу (Kali Linux).
hping3	–	Генератор/тестер мережевого трафіку й навантаження.
tcpdump	–	Консольний снифер/записувач PCAP.

ВСТУП

Актуальність дослідження. Стрімка популяризація Arduino- та ESP-платформ у проєктах Інтернету речей супроводжується зростанням кількості інцидентів на мережевому рівні. Обмежені обчислювальні ресурси мікроконтролерів, спрощені реалізації мережевих стеків (MQTT/HTTP/CoAP), типовість конфігурацій та відсутність формалізованих процедур тестування стійкості роблять такі вузли вразливими до базових атак, зокрема ARP-spoofing/MITM, SYN/UDP-flood, DNS-спуфінгу та повторного відтворення (replay). У реальних умовах це може призводити до відмови в обслуговуванні, некоректної роботи систем керування, втрати або підміни критично важливих даних.

Метою роботи є підвищення рівня захищеності та стійкості Arduino-базованих IoT-систем в умовах впливу мережевих атак шляхом розроблення комплексної методики оцінювання та впровадження ефективних механізмів захисту.

Об’єктом дослідження це процес функціонування та інформаційного обміну в локальних IoT-мережах на базі мікроконтролерів Arduino/ESP.

Предмет дослідження – методи, моделі та програмно-апаратні засоби виявлення й протидії мережевим атакам на Arduino-базовані IoT-системи, а також метрики оцінювання їх стійкості (доступність, цілісність, затримки передавання даних тощо).

Гіпотеза дослідження. Поєднання легковагових механізмів захисту (TLS-PSK, HMAC/nonce-підписування, обмеження швидкості обробки трафіку, ACL, сегментування мережі, правила IDS) дає змогу статистично значуще підвищити доступність сервісів і зменшити втрати та затримки під час атак без критичного впливу на продуктивність Arduino/ESP-вузлів.

Завдання дослідження:

- проаналізувати архітектуру Arduino/ESP, мережеві інтерфейси та протоколи взаємодії (MQTT/HTTP/CoAP) з позицій інформаційної безпеки;
- побудувати модель загроз та класифікацію базових мережевих атак для локальних IoT-мереж;
- спроєктувати експериментальний стенд (Arduino/ESP + датчики, брокер Mosquitto, атакувальна станція Kali Linux з Bettercap/hping3, IDS Suricata/Snort);
- сформувати систему метрик оцінювання стійкості (доступність, затримка, втрати, стабільність публікацій/підписок, показники виявлення атак IDS, MTTR);
- провести серію експериментів у режимах «без захисту» та «із захистом», забезпечивши відтворюваність умов і навантажень;
- виконати статистичний аналіз результатів, оцінити ефективність окремих і комплексних заходів захисту та розробити практичні рекомендації (шаблони конфігурацій, чек-листи).

Методи дослідження. У роботі використовуються експериментальне моделювання мережевих атак (ARP-spoofing/MITM, SYN/UDP-flood, DNS-спуфінг, replay), інструментальний аналіз трафіку (tcpdump, Wireshark), застосування систем виявлення вторгнень (Suricata/Snort), вимірювання продуктивності (RTT, втрати, показники QoS MQTT), а також статистичні методи обробки даних для перевірки значущості відмінностей між дослідними конфігураціями.

РОЗДІЛ 1.
АНАЛІЗ ЗАДАЧІ ЗАБЕЗПЕЧЕННЯ СТІЙКОСТІ ARDUINO-БАЗОВАНИХ IOT-СИСТЕМ

1.1. Сучасний стан розвитку технологій Інтернету речей та аналіз загроз безпеці

Сучасний етап еволюції інформаційно-комунікаційних технологій характеризується всеосяжною інтеграцією фізичних об'єктів у єдиний цифровий простір, що отримало назву Інтернет речей (Internet of Things, IoT). Ця парадигма стала технологічним фундаментом для концепцій «Індустрія 4.0», «Розумне місто» (Smart City) та систем точного землеробства [1]. За прогнозами провідних аналітичних агентств, зокрема IoT Analytics та Gartner, кількість підключених пристроїв у світі продовжує зростати експоненційно, перевищивши позначку в 15 мільярдів активних вузлів у 2023 році [2]. Таке зростання зумовлене здешевленням мікроелектронних компонентів та розвитком бездротових мереж низького енергоспоживання (LPWAN, ZigBee, Wi-Fi).

Особливістю поточного моменту є перехід від ізольованих вбудованих систем до розподілених мереж, де критичні рішення приймаються на основі даних, отриманих від тисяч сенсорів у режимі реального часу. Однак, масштабування IoT-мереж супроводжується пропорційним зростанням кіберзагроз. На відміну від традиційних корпоративних IT-систем, де периметр захисту чітко визначений, а обчислювальні ресурси серверів та робочих станцій дозволяють використовувати складні антивірусні засоби та фаєрволи, IoT-пристрої часто функціонують у недовіреному середовищі з мінімальним рівнем захисту.

Згідно зі звітами OWASP (Open Web Application Security Project), основними векторами атак на IoT є недостатня фізична безпека, використання слабких паролів за замовчуванням, відсутність механізмів оновлення прошивки та відсутність шифрування каналів зв'язку [3]. Стандарт NISTIR 8259A,

розроблений Національним інститутом стандартів і технологій США, визначає базовий набір можливостей кібербезпеки для IoT, серед яких ключовими є захист даних (Data Protection) та логічний контроль доступу (Access Control) [4]. Нехтування цими вимогами призводить до того, що бюджетні пристрої стають плацдармом для масштабних атак. Яскравим прикладом є ботнет Mirai, який у 2016 році об'єднав сотні тисяч вразливих камер відеоспостереження та роутерів для проведення найпотужнішої на той час DDoS-атаки, що паралізувала частину глобальної мережі Інтернет [5].

Теоретичний базис інформаційної безпеки в IoT спирається на класичну тріаду CIA (Confidentiality, Integrity, Availability), проте пріоритети в ній розставляються специфічно:

- **Доступність (Availability):** Для систем реального часу (керування виробництвом, моніторинг клімату) це критичний параметр. Затримка в передачі сигналу тривоги або команди на відключення насосу через DoS-атаку може призвести до фізичних збитків. Саме цей аспект є найбільш вразливим для мікроконтролерних систем через обмеженість їх ресурсів обробки трафіку [6].
- **Цілісність (Integrity):** Гарантія того, що дані не були модифіковані під час передачі. Підміна показників датчиків (наприклад, заниження температури в реакторі) може ввести систему автоматичного керування в оману.
- **Конфіденційність (Confidentiality):** Захист даних телеметрії від перехоплення. Хоча температура в кімнаті може здаватися неважливою інформацією, метадані (час присутності людей, режими роботи обладнання) можуть бути використані для планування фізичних злочинів.

Таким чином, забезпечення безпеки IoT вимагає комплексного підходу, що враховує специфічні вразливості та загрози, актуальні для гетерогенних мереж.

1.2. Архітектурні особливості та ресурсні обмеження платформ Arduino та ESP

При розробці системи захисту необхідно детально аналізувати апаратну специфіку об'єкта дослідження, оскільки саме вона диктує вибір методів захисту. У даній роботі розглядаються вузли, побудовані на базі платформи Arduino (мікроконтролери сімейства AVR) у поєднанні з комунікаційними модулями ESP8266/ESP32. Така архітектура є де-факто стандартом для прототипування, навчальних стендів та рішень малого бізнесу завдяки низькій вартості та розвиненій екосистемі, але вона має критичні обмеження.

Мікроконтролер ATmega328P, який є основою плати Arduino Uno, являє собою 8-бітний RISC-процесор із Гарвардською архітектурою, що працює на тактовій частоті 16 МГц. Його головним обмеженням у контексті реалізації алгоритмів безпеки є вкрай малий обсяг пам'яті: 32 КБ Flash-пам'яті для зберігання коду програми та всього 2 КБ оперативної пам'яті (SRAM) [7]. Оперативна пам'ять використовується для зберігання глобальних змінних, стеку викликів функцій та буферів вводу-виводу.

Реалізація сучасних криптографічних протоколів, таких як TLS 1.3 (Transport Layer Security), вимагає значних ресурсів. Наприклад, процес «рукостискання» (handshake) передбачає обмін цифровими сертифікатами стандарту X.509. Розмір навіть оптимізованого ланцюжка сертифікатів часто перевищує 2-4 КБ, що фізично не може бути розміщено в оперативній пам'яті ATmega328P. Спроби реалізувати TLS програмно призводять до переповнення стека (stack overflow) і перезавантаження контролера [8].

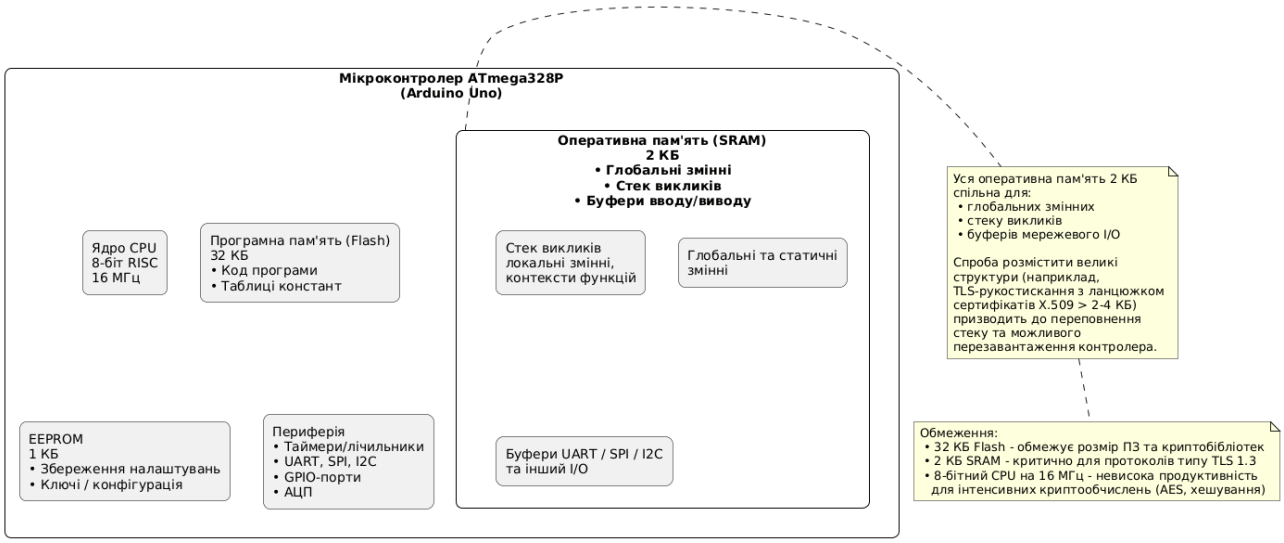

Рисунок 1.1 – Архітектура пам'яті та ресурсів мікроконтролера ATmega328P.

Для забезпечення мережевого підключення в такій архітектурі часто використовується модуль ESP8266 або ESP32, який виступає в ролі Wi-Fi модему. Взаємодія між керуючим контролером (Arduino) та мережевим процесором (ESP) відбувається через послідовний інтерфейс UART (Universal Asynchronous Receiver-Transmitter) за допомогою набору AT-команд [9].

Швидкість цього інтерфейсу стає «вузьким місцем» системи. Типова швидкість надійного з'єднання через програмний порт (SoftwareSerial) становить 9600 біт/с. Навіть при використанні апаратного UART на швидкості 115200 біт/с, пропускна здатність залишається низькою порівняно зі швидкістю Wi-Fi мережі (до 150 Мбіт/с).

Це створює специфічну проблему безпеки: якщо ESP8266 отримає пакет даних (наприклад, флуд-атаку), він спробує передати його на Arduino. Оскільки Arduino не встигає обробляти вхідний потік через повільний UART, внутрішні буфери переповнюються, що призводить до втрати легітимних керуючих команд і відмови в обслуговуванні (DoS) на рівні інтерфейсу.

Додатковим фактором є енергоспоживання. Криптографічні обчислення (хешування, шифрування AES) на 8-бітному процесорі вимагають великої кількості тактових циклів, що збільшує час активності процесора і, як наслідок, споживання енергії. Для пристроїв з батарейним живленням це є критичним обмеженням, яке змушує шукати компроміс між стійкістю алгоритму шифрування та часом автономної роботи.

1.3. Аналіз вразливостей мережевих протоколів та моделей атак

Функціонування IoT-систем забезпечується стеком протоколів TCP/IP, над яким працюють спеціалізовані протоколи прикладного рівня. Кожен рівень цієї моделі містить специфічні вразливості, які можуть бути використані зловмисниками.

Вразливості протоколу MQTT.

Протокол MQTT (Message Queuing Telemetry Transport), стандартизований OASIS, працює за моделлю «публікація-підписка» і є основним для IoT завдяки своїй легкості [10]. Проте базова специфікація v3.1.1 розроблялася для закритих мереж і має суттєві недоліки безпеки:

1. *Передача даних відкритим текстом:* У більшості стандартних реалізацій дані передаються через порт 1883 без шифрування. Використовуючи пасивний сніффінг (аналіз трафіку інструментами типу Wireshark або tcpdump), зловмисник може отримати доступ до всього інформаційного обміну, включаючи логіни та паролі, якщо вони передаються у payload.
2. *Слабка автентифікація:* Багато MQTT-брокерів (наприклад, Mosquitto) за замовчуванням налаштовані на прийом анонімних підключень (allow_anonymous true). Це дозволяє зловмиснику підключитися до системи, підписатися на wildcard-топік (#) і отримувати копію всіх повідомлень, або, що гірше, публікувати фальшиві команди у топіки керування приводами.

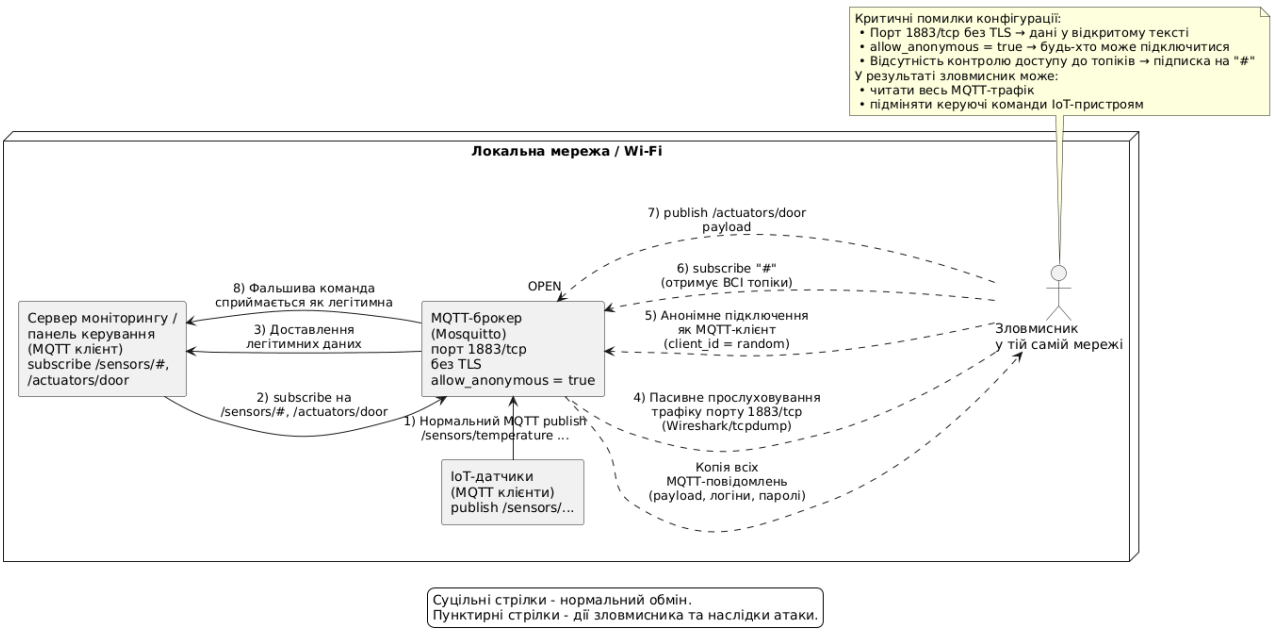

Рисунок 1.2 – Схема атаки через незахищений MQTT-брокер.

Вразливості канального рівня (ARP-Spoofing).

У локальних мережах Ethernet та Wi-Fi критичною загрозою є атака на протокол розв'язання адрес ARP (Address Resolution Protocol). Цей протокол відповідає за співставлення логічних IP-адрес із фізичними MAC-адресами пристроїв. Вразливість полягає у відсутності автентифікації: будь-який вузол мережі може надіслати відповідь на ARP-запит, навіть якщо його про це не питали (Gratuitous ARP).

Зловмисник, використовуючи інструменти на кшталт Bettercap або Ettercap, надсилає широкомовне повідомлення, в якому стверджує, що IP-адреса шлюзу (Gateway) відповідає його власній MAC-адресі. Одночасно він повідомляє шлюзу, що він є IoT-пристроєм. Це призводить до отруєння ARP-кешу (cache poisoning) жертв і дозволяє реалізувати атаку «Людина посередині» (Man-in-the-Middle, MITM). В результаті весь трафік проходить через вузол зловмисника, що дає можливість модифікувати дані «на льоту» або просто блокувати їх [11].

Атаки на відмову в обслуговуванні (DoS).

Для мікроконтролерів найнебезпечнішими є атаки, спрямовані на вичерпання ресурсів:

1. *UDP Flood:* Атакуючий генерує велику кількість UDP-дейтаграм на порт пристрою. Оскільки UDP є протоколом без встановлення з'єднання, атакувальник не обмежений необхідністю очікувати відповіді і може завантажити канал на повну потужність. Мікроконтролер змушений обробляти кожне переривання від мережевої карти, витрачаючи процесорний час на розбір заголовків сміттєвих пакетів, що призводить до повного «зависання» основної логіки програми.
2. *TCP SYN Flood:* Атака використовує особливості процедури «триетапного рукостискання» (three-way handshake) протоколу TCP. Атакувальник масово надсилає пакети з прапором SYN (запит на з'єднання), але ігнорує відповіді SYN-ACK від сервера. Вузол-жертва резервує пам'ять у таблиці з'єднань (TCB - Transmission Control Block) під ці напіввідкриті з'єднання. Оскільки пам'ять Arduino/ESP вкрай обмежена, таблиця миттєво заповнюється, і нові легітимні підключення відхиляються [12].

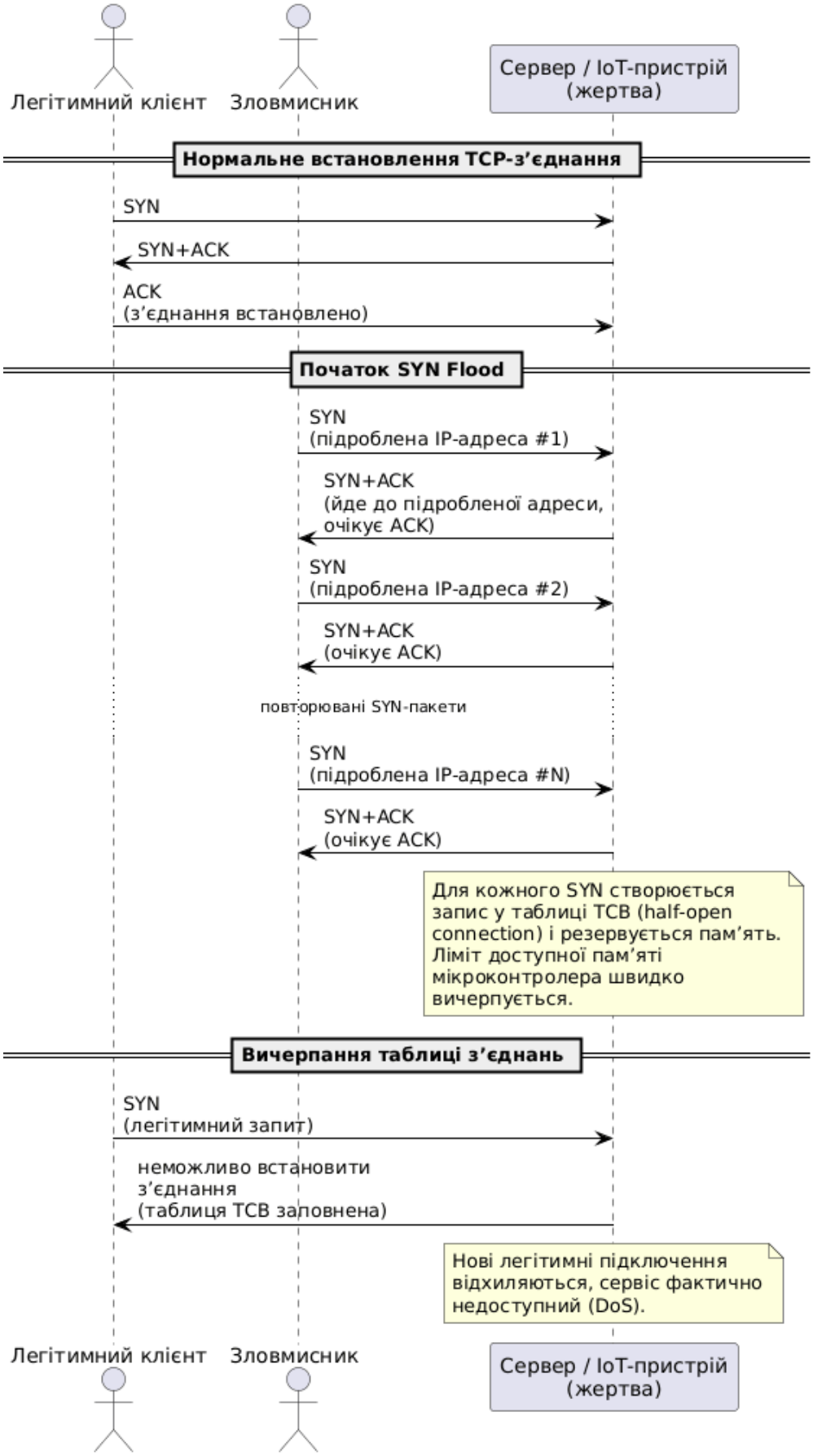

Рисунок 1.3 – Часова діаграма атаки TCP SYN Flood

Також актуальними є **атаки повторного відтворення (Replay Attacks)**. Навіть якщо дані зашифровані або підписані, зловмисник може перехопити валідний пакет (наприклад, команду «відкрити замок») і надіслати його повторно через деякий час. Без механізмів перевірки свіжості даних (timestamp або nonce) система виконає команду повторно, що є порушенням безпеки.

1.4. Огляд існуючих методів та засобів забезпечення мережевої безпеки IoT

Для протидії зазначеним загрозам розроблено широкий спектр методів захисту, які можна класифікувати на криптографічні (рівень даних) та інфраструктурні (рівень мережі).

Криптографічний захист: дилема TLS та HMAC.

«Золотим стандартом» безпеки в Інтернеті є протокол TLS (Transport Layer Security) [13]. Він забезпечує повний комплекс захисту: конфіденційність (шифрування), цілісність (MAC) та автентифікацію (сертифікати). Однак для 8-бітних систем він є надлишковим. Існують спрощені бібліотеки (наприклад, BearSSL, axtls), але вони вимагають компромісів: використання слабших шифрів або відмову від перевірки сертифікатів, що робить систему вразливою до MITM. Крім того, час встановлення з'єднання TLS на ESP8266 може досягати кількох секунд, що є неприйнятним для деяких застосувань.

Альтернативою є використання HMAC (Hash-based Message Authentication Code) — механізму автентифікації повідомлень за допомогою хеш-функцій [14]. Суть методу полягає в тому, що до повідомлення додається цифровий підпис, отриманий шляхом хешування (наприклад, алгоритмом SHA-256) самого повідомлення разом із секретним ключем.

Переваги HMAC для Arduino:

- *Швидкодія:* Обчислення SHA-256 на 8-бітному процесорі займає мілісекунди, на відміну від асиметричної криптографії (RSA/ECC), яка може займати секунди [15].

- *Компактність:* Реалізація HMAC займає мінімум Flash-пам'яті і не потребує великих буферів в RAM.
- *Гнучкість:* HMAC можна використовувати поверх будь-якого протоколу (HTTP, MQTT, UDP).

Головний недолік HMAC — відсутність шифрування тіла повідомлення (дані видно, але не можна змінити) та необхідність попереднього розподілу ключів (Pre-Shared Keys).

Мережеві системи виявлення вторгнень (IDS).

Системи IDS (Intrusion Detection Systems), такі як Suricata або Snort, дозволяють аналізувати копію мережевого трафіку на предмет відповідності сигнатурам відомих атак [16]. У контексті IoT IDS може бути налаштована на виявлення специфічних аномалій:

- Поява нових MAC-адрес у сегменті мережі.
- Виявлення пакетів ARP з конфліктуючими адресами (ознака спуфінгу).
- Перевищення порогу частоти запитів від одного пристрою (ознака DoS).

Інтеграція IDS дозволяє реалізувати захист без модифікації прошивки IoT-пристроїв, переклавши задачу аналізу на шлюз або окремий сервер.

Механізми обмеження навантаження (Rate Limiting).

Для захисту від DoS-атак ефективним є застосування алгоритмів обмеження швидкості (Rate Limiting). Вони можуть бути реалізовані на двох рівнях:

1. *На стороні брокера/сервера:* Обмеження кількості повідомлень за секунду від одного ClientID.
2. *На стороні пристрою:* Алгоритм «Token Bucket» (відро з маркерами) дозволяє пристрою контролювати частоту відправки повідомлень і

ігнорувати вхідні запити, якщо їх частота перевищує можливості обробки, запобігаючи переповненню буфера [17].

Висновки до першого розділу

У першому розділі проаналізовано сучасні IoT-системи на базі Arduino/ESP Проведений аналіз показує, що існує суттєвий розрив між теоретичними рекомендаціями щодо безпеки IoT та практичними можливостями бюджетних апаратних платформ. Стандартні «важкі» протоколи блокують роботу мікроконтролера, а повна відсутність захисту робить систему легкою здобиччю для зловмисників.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

1. Проаналізувати архітектуру та вразливості стеку протоколів Arduino/ESP, виявити критичні обмеження ресурсів.
2. Побудувати модель загроз та класифікацію атак для локальних IoT-мереж (ARP-spoofing, SYN/UDP flood, MQTT replay).
3. Спроєктувати та розгорнути експериментальний стенд, що включає IoT-вузли, брокер повідомлень Mosquitto, атакувальну станцію (Kali Linux) та систему захисту (IDS Suricata).
4. Розробити та програмно реалізувати комплекс контрзаходів: алгоритм автентифікації на базі HMAC-SHA256 з захистом від повторів (anti-replay nonce) та правила фільтрації трафіку.
5. Провести серію експериментів для порівняльної оцінки ефективності захисту за метриками доступності (успішність доставки пакетів), часових затримок та стабільності роботи під навантаженням.
6. Сформувати практичні рекомендації (чек-листи, шаблони коду) для налаштування безпечних Arduino-систем.

РОЗДІЛ 2.
МОДЕЛЮВАННЯ ЗАГРОЗ ТА ПРОЕКТУВАННЯ СИСТЕМИ ЗАХИСТУ IOT-МЕРЕЖІ

2.1. Архітектура та технічне забезпечення експериментального стенду

Для проведення експериментального дослідження стійкості IoT-систем до мережевих атак та верифікації запропонованих методів захисту було спроектовано та реалізовано спеціалізований програмно-апаратний стенд. Основною метою створення стенду є забезпечення контрольованого та відтворюваного середовища, яке дозволяє моделювати сценарії експлуатації вразливостей, характерні для реальних систем «розумного дому» (Smart Home) та промислового Інтернету речей (IIoT), без ризику порушення роботи діючої інфраструктури.

Архітектура досліджуваної мережі побудована за топологією «зірка» (Star Topology) і базується на стандартах сімейства IEEE 802.11 (Wi-Fi). Логічно структура стенду розділена на три функціональні зони, що відповідають ролям учасників мережевої взаємодії: зона вразливих пристроїв (Victim Zone), зона керування та моніторингу (Management Zone) та зона генерації загроз (Attacker Zone). Усі вузли об'єднані в єдиний широкомовний домен (Broadcast Domain) через бездротовий маршрутизатор, що створює умови для реалізації атак канального рівня.

Апаратна складова стенду

1. Кінцевий вузол IoT (Жертва / Victim).

В якості фізичної моделі типового низькоресурсного IoT-пристрою обрано платформу Arduino Uno Rev3. Вибір саме цієї плати обумовлений необхідністю дослідження граничних можливостей захисту на апаратурі з критичними обмеженнями обчислювальної потужності.

- *Обчислювальне ядро:* Мікроконтролер ATmega328P (архітектура AVR, розрядність 8 біт, тактова частота 16 МГц). Обсяг оперативної пам'яті (SRAM) становить лише 2 КБ, що є основним обмежуючим фактором для реалізації криптографічних протоколів [1].

- *Комунікаційний модуль:* Для забезпечення бездротового зв'язку використано модуль **ESP8266 (модифікація ESP-01)**, побудований на базі 32-бітного процесора Tensilica L106. Модуль функціонує в режимі «AT-модему», виконуючи роль мережевого моста (UART-to-Wi-Fi).

- *Інтерфейс взаємодії:* Фізичне з'єднання між Arduino та ESP8266 реалізовано через піни D2 (RX) та D3 (TX) з використанням програмної бібліотеки SoftwareSerial. Швидкість обміну даними обмежена на рівні 9600 біт/с для забезпечення стабільності програмного UART. Таке архітектурне рішення вводить штучне «вузьке місце» (bottleneck), що дозволяє дослідити поведінку системи при виникненні черг повідомлень під час DoS-атак [2].

- *Сенсорна підсистема:* Джерелом даних телеметрії виступає цифровий датчик температури та вологості **DHT11**, підключений до цифрового входу D4.

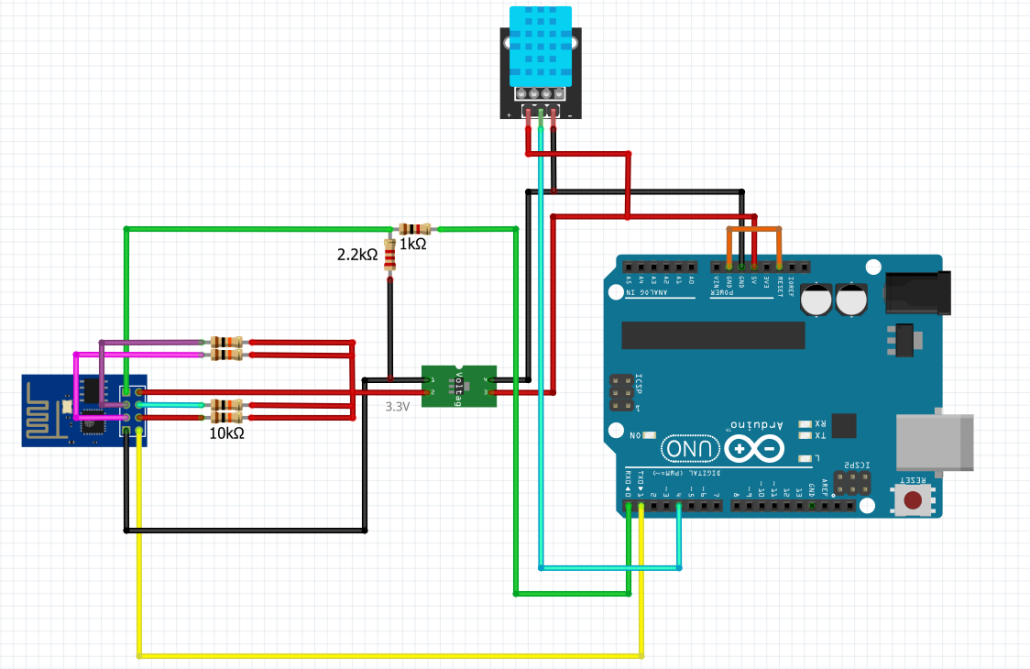

Рисунок 2.1 – Апаратна конфігурація кінцевого IoT-вузла (Arduino Uno + ESP8266).

2. Сервер інфраструктури (Брокер / Broker & IDS).

Центральний вузол мережі розгорнуто на базі одноплатного мікрокомп'ютера Raspberry Pi 4 Model B (4 ГБ RAM, 4-ядерний процесор ARM Cortex-A72). Цей пристрій емулює хмарний сервер або локальний шлюз IoT. На ньому розгорнуто сервіси:

- MQTT-брокер (Eclipse Mosquitto);
- База даних часових рядів (InfluxDB) для збереження телеметрії;
- Система виявлення вторгнень (Suricata IDS) для пасивного аналізу трафіку.

Висока продуктивність цього вузла дозволяє виключити затримки на стороні сервера з загального балансу часу реакції системи.

3. Станція зловмисника (Атакуючий / Attacker).

Для генерації тестових атак використовується ноутбук під керуванням спеціалізованої операційної системи Kali Linux (версія 2024.1). Мережевий адаптер станції переведено в режим моніторингу (monitor mode), що дозволяє здійснювати перехоплення (sniffing) та ін'єкцію (injection) довільних пакетів у бездротове середовище. Станція містить набір інструментів для пентестингу: Bettercap, hping3, Wireshark, Metasploit Framework [3].

4. Комутаційне обладнання.

Маршрутизатор TP-Link TL-WR841N, налаштований на роботу в стандарті 802.11n (2.4 ГГц) з шириною каналу 20 МГц. Використовується шифрування WPA2-PSK для імітації захищеної домашньої мережі, проте це не захищає від атак всередині периметра (Internal Threats).

Програмна складова та середовище розробки

Програмна реалізація прошивки мікроконтролера виконана мовою C++ у середовищі **Arduino IDE 2.3.2**. Для роботи з периферією використано наступні бібліотеки:

- SoftwareSerial.h – для емуляції UART-порту;
- WiFiEsp.h – драйвер для керування модулем ESP8266 AT-командами;
- PubSubClient.h – клієнтська реалізація протоколу MQTT;
- DHT.h – драйвер сенсора DHT11;
- sha256.h – криптографічна бібліотека для реалізації алгоритму HMAC (буде детально розглянуто в п. 2.4).

На серверній стороні (Raspberry Pi) використовується ОС **Ubuntu Server 22.04 LTS**. Контейнеризація сервісів реалізована за допомогою **Docker**, що забезпечує швидке розгортання та ізоляцію компонентів (Broker, IDS, Dashboard).

Схема інформаційних потоків у стенді виглядає наступним чином:

1. Сенсор DHT11 генерує цифрові дані (T, H).
2. Arduino зчитує дані, формує пакет (JSON), додає підпис (HMAC) і передає його через SoftwareSerial на ESP8266.
3. ESP8266 інкапсулює дані в TCP/IP пакети і передає їх через Wi-Fi на роутер.
4. Роутер маршрутизує пакети до Raspberry Pi (Брокера).
5. Брокер перевіряє валідність пакету та публікує його для підписників.
6. Паралельно IDS Suricata аналізує копію трафіку на наявність сигнатур атак.

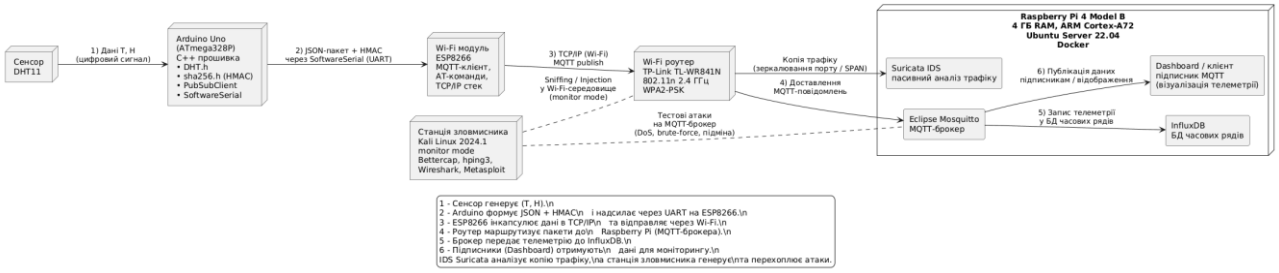

Рисунок 2.2 – Структурна схема інформаційної взаємодії компонентів експериментального стенду

Така архітектура дозволяє проводити вимірювання на кожному етапі проходження сигналу, локалізувати джерела затримок та оцінювати вплив атак на конкретні вузли системи.

2.2. Обґрунтування вибору програмних засобів та інструментів дослідження

Ефективність проведення експериментального дослідження та достовірність отриманих результатів значною мірою залежать від коректного підбору програмного інструментарію. У даній роботі використано комплекс

програмних засобів з відкритим вихідним кодом (Open Source), що забезпечує прозорість методології та можливість відтворення експерименту іншими дослідниками. Вибір інструментів базувався на критеріях функціональності, підтримки спільнотою та відповідності специфіці IoT-протоколів.

1. Протокол прикладного рівня MQTT

Для організації обміну даними в експериментальній мережі обрано протокол MQTT (Message Queuing Telemetry Transport) версії 3.1.1, стандартизований консорціумом OASIS [4].

На відміну від традиційного протоколу HTTP, який працює за моделлю «запит-відповідь» і має значний розмір заголовків (від сотень байт), MQTT використовує компактний бінарний формат повідомлень та асинхронну модель «публікація-підписка» (Publish/Subscribe). Це дозволяє:

- Мінімізувати накладні витрати трафіку (overhead), що критично для низькошвидкісного каналу UART (9600 біт/с).
- Забезпечити роботу пристроїв у мережах з нестабільним з'єднанням (завдяки механізмам QoS та Retain).
- Зменшити енергоспоживання мікроконтролера за рахунок скорочення часу активності радіомодуля.

В якості брокера повідомлень використано **Eclipse Mosquitto**. Це кросплатформене рішення, написане мовою C, яке характеризується високою продуктивністю та низьким споживанням пам'яті. Mosquitto підтримує необхідні для дослідження механізми безпеки:

- Автентифікацію клієнтів за ім'ям користувача та паролем;
- Списки контролю доступу (ACL) для розмежування прав на читання/запис топіків;
- Шифрування TLS (використовується для порівняльного аналізу).

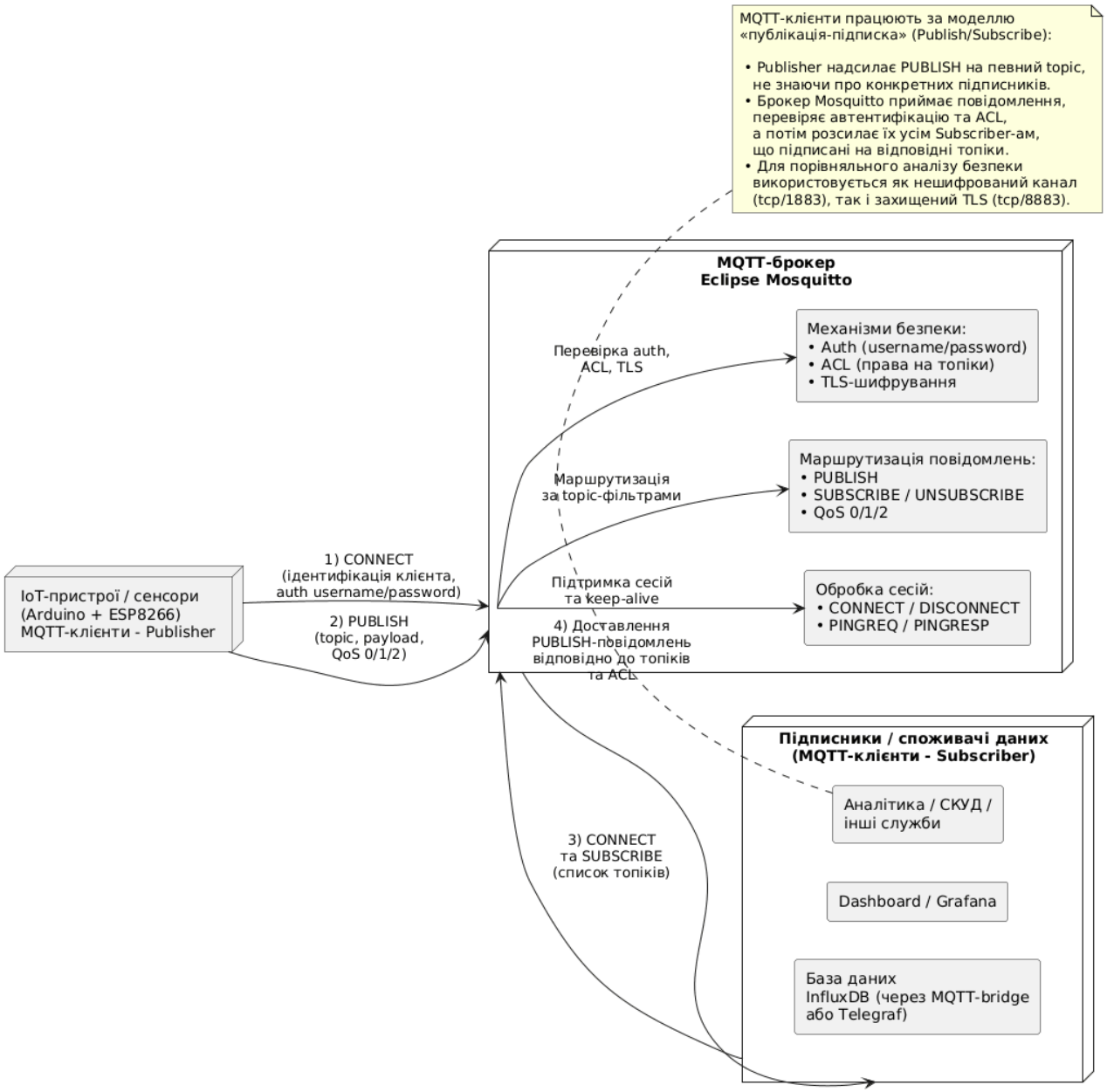

Рисунок 2.3 – Модель взаємодії компонентів протоколу MQTT.

2. Засоби аналізу та генерації атак (Attacker Toolkit)

Для моделювання дій зловмисника на станції Kali Linux використано спеціалізовані утиліти, які дозволяють реалізувати сценарії атак на різних рівнях моделі OSI:

- **Bettercap (v2.32):** Універсальний інструмент для проведення атак типу Man-in-the-Middle (MITM). На відміну від застарілих аналогів

(Ettercap), Bettercap має модульну архітектуру та дозволяє скриптувати атаки на мові JavaScript. У роботі він використовується для:

- Сканування мережі та ідентифікації пристроїв (Network Reconnaissance).
- Виконання ARP Spoofing для перенаправлення трафіку жертви.
- Перехоплення та модифікації (Tampering) MQTT-пакетів "на льоту" за допомогою модуля tcp.proxy.

- **Hping3:** Генератор пакетів TCP/IP, що дозволяє конструювати довільні пакети з заданими прапорами та полями заголовка. Використовується для стрес-тестування мережевого стека Arduino шляхом моделювання атак:
  - *TCP SYN Flood:* Генерація пакетів із прапором SYN для вичерпання таблиці з'єднань.
  - UDP Flood: Насичення каналу великою кількістю сміттєвих даних.

Основною перевагою hping3 є можливість точного налаштування частоти відправки пакетів (опція --fast, --flood), що дозволяє визначити поріг відмови обслуговування (DoS Threshold) досліджуваної системи [5].

- **Wireshark (v4.2.0):** Стандартний інструмент для захоплення та глибокого аналізу мережевих пакетів. Використовується для верифікації проходження атак, вимірювання часових затримок (Round Trip Time, RTT) та візуалізації вмісту пакетів до та після впровадження шифрування. Спеціалізований дисектор MQTT у Wireshark дозволяє розбирати структуру повідомлень (Fixed Header, Variable Header, Payload).

3. Система виявлення вторгнень (IDS)

Для захисту на мережевому рівні обрано IDS Suricata. Це високопродуктивна система виявлення загроз нового покоління, яка має ряд переваг перед класичним Snort:

- *Багатопотоковість (Multi-threading):* Ефективне використання ресурсів багатоядерного процесора Raspberry Pi, що дозволяє обробляти трафік на швидкості до 10 Гбіт/с (в лабораторних умовах це гарантує відсутність втрат пакетів при аналізі).
- *Підтримка прикладних протоколів:* Вбудовані парсери для HTTP, DNS, TLS та MQTT, що дозволяє писати правила фільтрації, орієнтовані на вміст полів протоколу, а не лише на порти та IP-адреси.
- *Режим IPS:* Можливість роботи в режимі запобігання (Inline), коли система не просто сигналізує про атаку, а автоматично блокує шкідливий трафік через інтеграцію з фаєрволом NFQUEUE.

4. Середовище візуалізації та збору даних

Для моніторингу параметрів системи в реальному часі використано стек TIG (Telegraf + InfluxDB + Grafana).

- *Telegraf:* Агент, що збирає метрики з брокера Mosquitto (кількість підключень, обсяг трафіку) та системні метрики Raspberry Pi (CPU load, RAM usage).
- *InfluxDB:* База даних часових рядів (Time Series Database), оптимізована для збереження великих обсягів метрик.
- *Grafana:* Платформа для візуалізації, яка дозволяє будувати дашборди для відображення динаміки атак (наприклад, графік "Кількість MQTT-пакетів за секунду" дозволяє миттєво побачити початок DoS-атаки).

Такий набір інструментів дозволяє охопити весь цикл дослідження: від генерації контрольованого впливу до детального аналізу реакції системи та візуалізації результатів.

2.3 Математичне та логічне моделювання мережевих атак

Для формалізації процесу тестування та отримання об'єктивних кількісних оцінок стійкості необхідно побудувати математичні та логічні моделі загроз. Це дозволяє абстрагуватися від конкретних програмних реалізацій інструментів атаки (Bettercap, hping3) і визначити універсальні критерії успішності атаки або захисту. У даній роботі розглядаються моделі трьох класів атак, найбільш критичних для досліджуваної архітектури: порушення маршрутизації (MITM), відмова в обслуговуванні (DoS) та повторне відтворення (Replay).

2.3.1 Моделювання атаки arp spoofing (mitm)

Атака «Людина посередині» на канальному рівні (Layer 2 OSI) базується на маніпуляції протоколом розв'язання адрес ARP (Address Resolution Protocol). Нехай локальна мережа N складається з множини вузлів $H = \{V, A, G\}$, де V (Victim) — IoT-пристрій (Arduino), A (Attacker) — вузол зловмисника, G (Gateway) — шлюз. Функціонування протоколу ARP можна описати як функцію відображення логічних адрес у фізичні [18, 27]:

$$f_{ARP}: IP \rightarrow MAC$$

У нормальному стані роботи мережі таблиця ARP (кеш) вузла-жертви містить коректний запис для шлюзу [18, 28]:

$$ARP\_Table_V(IP_G) = MAC_G$$

Модель атаки полягає у навмисному спотворенні цієї функції відображення. Зловмисник генерує потік відповідей R_{arp} з інтенсивністю, що перевищує час життя запису в кеші, стверджуючи, що IP-адреса шлюзу належить його MAC-адресі. В результаті успішної атаки таблиця жертви змінюється [18, 28]:

$$ARP\_Table_V(IP_G) = MAC_A$$

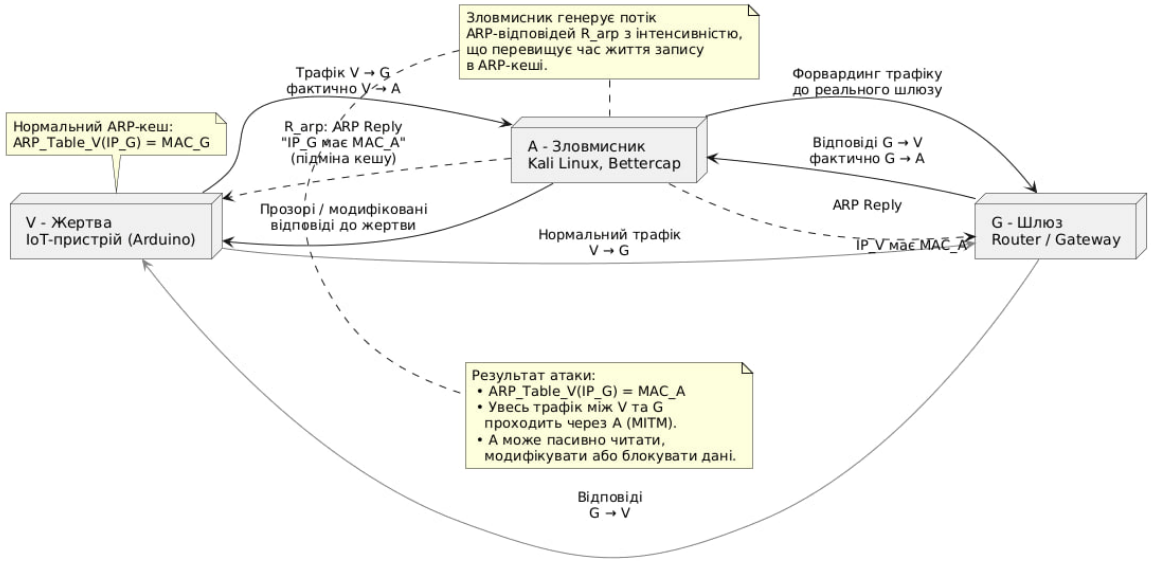

Рисунок 2.4 – Граф перенаправлення потоків даних при атаці ARP Spoofing.

Формальним критерієм успішності атаки MITM є ймовірність перехоплення трафіку, яка в ідеальних умовах атаки має дорівнювати одиниці [18, 28]:

$$P(Traffic_{V \to G} \cap Interface_A) = 1$$

Виконання цієї умови надає зловмиснику можливість реалізувати функцію модифікації повідомлення M у повідомлення M' перед пересиланням його шлюзу [18, 28]:

$$M' = F_{tamper}(M)$$

Якщо система не використовує перевірку цілісності (HMAC), то $Verify(M') = True$, тобто атака на цілісність даних є успішною.

2.3.2 Моделювання атаки denial of service (dos) на ресурс мікроконтролера

Атака на пристрої класу Arduino/ESP моделюється не як класичне переповнення пропускної здатності каналу, а як задача теорії масового

обслуговування (ТМО) з обмеженою чергою. Розглянемо IoT-пристрій як систему масового обслуговування типу **M/M/1/N**, де:

- Вхідний потік заявок (мережевих пакетів) має інтенсивність λ (пакетів/с);
- Продуктивність (швидкість обробки) мікроконтролера — μ (пакетів/с);
- Розмір буфера вхідної черги (SoftwareSerial buffer) — N.

Коефіцієнт завантаження системи визначається як відношення інтенсивності вхідного потоку до продуктивності [27]:

$$\rho = \frac{\lambda}{\mu}$$

Умова стабільної роботи системи вимагає, щоб $\rho < 1$. Під час атаки (наприклад, UDP Flood або TCP SYN Flood) зловмисник штучно збільшує інтенсивність потоку до значень λ_{attack}, що значно перевищують можливості обробки пристроєм [23, 26]:

$$\lambda_{attack} \gg \mu$$

Ймовірність втрати пакету (P_{loss}) через переповнення буфера розміром N (для стандартної бібліотеки Arduino SoftwareSerial $N = 64$ байти) обчислюється за формулою для системи з відмовами [27]:

$$P_{loss} = \frac{\rho^N(1-\rho)}{1-\rho^{N+1}}$$

При інтенсивній атаці, коли $\rho \to \infty$, ймовірність втрати пакетів прагне до одиниці ($P_{loss} \to 1$), що означає повну недоступність сервісу.

Окрім переповнення буфера, критичним фактором є навантаження на центральний процесор (CPU) через обробку апаратних переривань. Якщо час

обробки одного переривання (прийом байту) становить t_{int}, то сумарне завантаження CPU атакуючим трафіком становить [23, 26]:

$$Load_{CPU} = \lambda_{attack} \cdot Size_{packet} \cdot t_{int}$$

Коли $Load_{CPU}$ наближається до 1 (100%), основний цикл програми `loop()` блокується, і пристрій перестає опитувати датчики та керувати виконавчими механізмами.

2.3.3 Моделювання атаки повторного відтворення (replay attack)

Атака повторного відтворення спрямована на порушення актуальності даних або несанкціоноване виконання команд. Нехай повідомлення M_i складається з корисних даних та тегу автентифікації (або статичного пароля) [13, 21]:

$$M_i = \langle Data, AuthTag \rangle$$

У вразливій системі умова валідності перевіряє лише коректність підпису/пароля, що дозволяє зловмиснику повторно надіслати раніше перехоплене старе повідомлення M_i у будь-який момент часу t [13, 21]:

$$\forall t: Verify(M_i, t) = True$$

Для захисту від цієї загрози вводиться параметр унікальності η (nonce — number used once), що монотонно зростає з кожним новим повідомленням, або мітка часу. Модель захищеного повідомлення набуває вигляду [16, 29]:

$$M_i = \langle Data, \eta_i, HMAC(K, Data||\eta_i) \rangle$$

Тоді функція верифікації на стороні сервера перетворюється на систему логічних умов [16, 29]:

$$Verify(M_i) = \begin{cases} True, & if\ HMAC_{valid} \wedge \eta_i > \eta_{last\_seen} \\ False, & otherwise \end{cases}$$

Таким чином, математично задача захисту від Replay-атак зводиться до забезпечення надійної синхронізації та перевірки свіжості параметра η. Будь-яке

повідомлення з $\eta \leq \eta_{last\_seen}$ буде відхилено, навіть якщо криптографічний підпис є вірним.

2.4. Проектування комплексної системи захисту та алгоритмів автентифікації

Враховуючи обмежені обчислювальні ресурси мікроконтролера ATmega328P, реалізація повноцінного стека TLS/SSL є недоцільною через високі накладні витрати пам’яті та часу процесора. Тому для забезпечення цілісності та автентичності даних розроблено полегшений протокол прикладного рівня на базі HMAC (Hash-based Message Authentication Code). Для захисту від атак на відмову в обслуговуванні (DoS) на рівні кінцевого пристрою спроектовано алгоритм обмеження швидкості (Rate Limiting), а для захисту інфраструктури — набір сигнатур для IDS Suricata.

2.4.1 Проектування алгоритму автентифікації hmac-sha256 з захистом від повторів

Розроблений протокол передбачає додавання цифрового підпису до кожного MQTT-повідомлення. Структура захищеного пакету (payload) визначається у форматі JSON наступним чином:

{

"d": { "temp": 24.5, "hum": 60 }, // Корисне навантаження (Data)

"n": 1054, // Унікальний лічильник (Nonce)

"ts": 1698745200, // Мітка часу (Timestamp)

"s": "a1b2c3d4..." // Підпис HMAC (Signature)

}

Алгоритм формування підпису (на стороні Arduino):

1. **Формування рядка для підпису (Str_{sign}):** Конкатенація значень полів даних, nonce та timestamp у суворо визначеному порядку без розділювачів.

$$Str_{sign} = String(Data) + String(Nonce) + String(Timestamp)$$

2. **Хешування:** Обчислення HMAC за допомогою алгоритму SHA-256 з використанням попередньо узгодженого секретного ключа (K_{secret}).

$$Signature = HMAC\_SHA256(K_{secret}, Str_{sign})$$

3. **Інкрементація Nonce:** Збільшення лічильника $Nonce$ на 1 та збереження його в енергонезалежній пам’яті (EEPROM) для збереження стану при перезавантаженні.

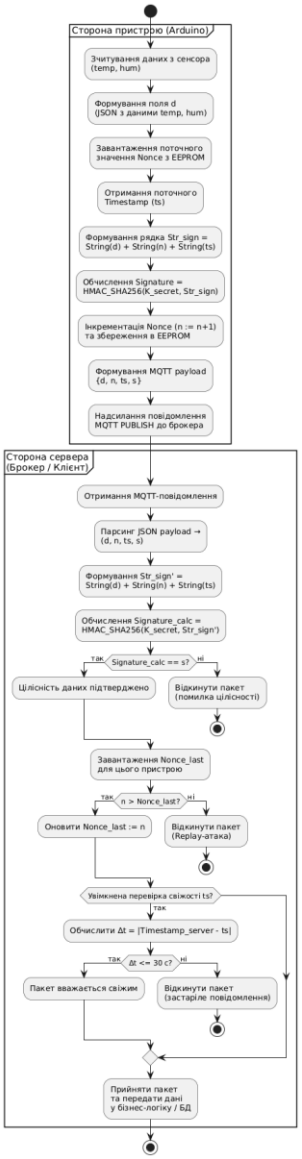

Рисунок 2.5 – Блок-схема алгоритму формування та перевірки підпису HMAC.

Алгоритм верифікації (на стороні Брокера/Клієнта):

Сервер отримує повідомлення, витягує з нього Data, Nonce, Timestamp, Signature і виконує перевірку за трьома критеріями:

1. **Перевірка цілісності:** Сервер самостійно обчислює $Signature_{calc}$ від отриманих даних та свого ключа. Якщо $Signature_{calc} \neq Signature$, пакет відкидається (спроба підміни даних).

2. **Захист від повторів (Anti-Replay):** Сервер зберігає останнє отримане значення $Nonce_{last}$ для кожного пристрою. Якщо $Nonce_{received} \leq Nonce_{last}$, пакет відкидається (атака повторного відтворення).

3. **Перевірка свіжості (опціонально):** Якщо різниця $|Timestamp_{server} - Timestamp_{device}|$ перевищує порогове значення (наприклад, 30 секунд), пакет вважається застарілим.

2.4.2 Проектування механізму обмеження швидкості (token bucket)

Для захисту мікроконтролера від зависання під час інтенсивних DoS-атак (коли пристрій витрачає весь час на обробку переривань від мережевої карти) розроблено програмний фільтр на основі алгоритму **Token Bucket** («Маркерне відро»).

Суть методу полягає в тому, що пристрій має віртуальний «бюджет» на обробку пакетів.

- C (Capacity) — ємність відра (максимальна кількість пакетів, які можна обробити миттєво/в черзі).
- R (Refill Rate) — швидкість поповнення бюджету (токени/сек).

Логіка роботи алгоритму (Псевдокод для Arduino):

1. При кожному виклику циклу `loop()` оновлюється кількість токенів [23, 27]:

$$Tokens = \min(C, Tokens + (Time_{now} - Time_{last}) \cdot R)$$

2. При надходженні вхідного пакету через UART перевіряється наявність токенів.

3. **Умова фільтрації:**

- Якщо $Tokens \geq 1$: пакет передається на розбір (JSON parsing) та перевірку підпису. Віднімається 1 токен.

- Якщо $Tokens < 1$: пакет ігнорується (скидається буфер UART), ресурсоємні функції не викликаються.

Такий підхід дозволяє гарантувати, що навіть при штормі в 1000 пакетів/сек, мікроконтролер буде обробляти лише R пакетів/сек (наприклад, 5), зберігаючи ресурси для підтримки з'єднання з мережею та опитування датчиків.

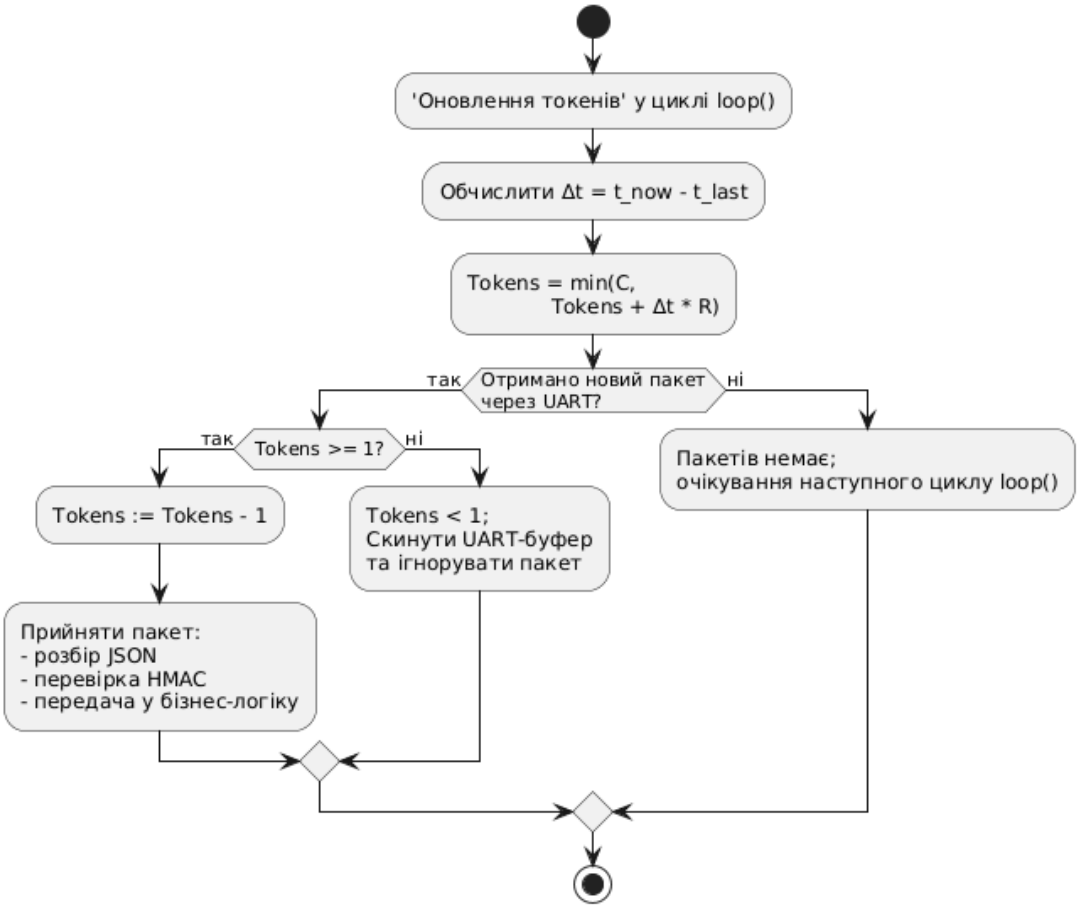

Рисунок 2.6 – Принцип роботи алгоритму Token Bucket для фільтрації трафіку.

2.4.3 Розробка правил фільтрації для ids suricata

Для виявлення атак на мережевому рівні та автоматичного блокування зловмисників спроектовано набір сигнатур (rules) для системи виявлення вторгнень Suricata. Правила орієнтовані на специфіку протоколів ARP та MQTT.

Правило 1: Виявлення ARP Spoofing (Конфлікт MAC-адрес) Правило відстежує ARP-пакети, в яких IP-адреса шлюзу (наприклад, 192.168.1.1) оголошується з MAC-адресою, відмінною від відомої адреси маршрутизатора.

alert arp any any -> any any (msg:"SECURITY: ARP Spoofing Attempt Detected"; \

content:"|08 06|"; \

content:"|00 02|"; depth:2; offset:20; \

sip:192.168.1.1; \

mac-address:!aa:bb:cc:dd:ee:ff; \

classtype:protocol-command-decode; sid:1000001; rev:1;)

Де aa:bb:cc:dd:ee:ff — легітимна MAC-адреса шлюзу.

Правило 2: Виявлення DoS-атаки на MQTT (Connection Flood) Правило виявляє аномально високу частоту спроб встановлення з’єднання з брокером (пакети MQTT CONNECT) з однієї IP-адреси.

alert tcp any any -> $HOME_NET 1883 (msg:"DOS: MQTT Connection Flood"; \

flow:to_server; \ content:"|10|"; depth:1; \

threshold: type both, track by_src, count 10, seconds 1; \

classtype:denial-of-service; sid:1000002; rev:1;)

Це правило спрацює, якщо один клієнт надішле більше 10 запитів на підключення протягом 1 секунди.

Правило 3: Виявлення спроб публікації без авторизації Правило аналізує заголовки MQTT пакетів на наявність спроб публікації в критичні топіки (наприклад, `/admin/commands`) без валідного HMAC-підпису в payload (перевірка наявності ключа `"s":`).

alert tcp any any -> $HOME_NET 1883 (msg:"SECURITY: Potential Unauthorized MQTT Publish"; \

flow:to_server; \

content:"PUBLISH"; \

content:"/admin/commands"; \

pcre:"!/\"s\":\"[a-f0-9]{64}\"/"; \

classtype:attempted-admin; sid:1000003; rev:1;)

Запропонований комплекс заходів (HMAC + Token Bucket + IDS) створює ешелоновану систему захисту, яка перекриває вектори атак на рівнях пристрою, протоколу та мережевої інфраструктури.

2.5. Методика проведення експериментів та метрики оцінювання

Для отримання об’єктивних даних про ефективність запропонованих рішень розроблено методику тестування, яка передбачає проведення серії експериментів за сценарієм «до» та «після» впровадження захисту. Експериментальне дослідження поділяється на три логічні фази, що дозволяє порівняти поведінку системи у різних станах.

Сценарій тестування

1. **Фаза 1: Baseline (Базова лінія).** Система функціонує у штатному режимі без зовнішнього втручання.

- *Мета:* Визначити еталонні показники продуктивності (затримки, стабільність з'єднання) для "чистої" мережі.
- *Навантаження:* IoT-пристрій надсилає телеметрію з інтервалом $T = 1s$.
- *Тривалість:* 600 секунд (10 хвилин).

2. **Фаза 2: Attack Phase (Під навантаженням).** Активація генераторів атак на станції зловмисника.
   - *Сценарій А (DoS):* Запуск `hping3` у режимі UDP Flood з інтенсивністю, що зростає ступінчасто (від 10 до 100 пакетів/с).
   - *Сценарій Б (MITM):* Запуск `bettercap` для ARP-спуфінгу та перехоплення трафіку.
   - *Мета:* Зафіксувати момент деградації сервісу (точка відмови) та виміряти втрати.
3. **Фаза 3: Defense Phase (Захист активовано).** Увімкнення розроблених механізмів (HMAC-підпис, Rate Limiting у прошивці, правила IDS). Повторення атак з Фази 2.
   - *Мета:* Оцінити здатність системи протистояти атакам та виміряти накладні витрати ресурсів (overhead), спричинені роботою алгоритмів захисту.

Метрики оцінювання ефективності

Для кількісного порівняння результатів використовуються наступні метрики:

1. **Коефіцієнт доставки пакетів (Packet Delivery Ratio, PDR).** Основний індикатор доступності системи. Визначається як відношення кількості повідомлень, успішно отриманих та верифікованих брокером (N_{rx}), до загальної кількості відправлених повідомлень (N_{tx}) [23, 26]:

$$PDR = \left(\frac{N_{rx}}{N_{tx}}\right) \cdot 100\%$$

В умовах успішної DoS-атаки цей показник прямує до 0. Ефективний захист має утримувати PDR на рівні > 90%.

2. **Середня затримка передачі (Latency / RTT).** Час повного обороту пакету, що вимірюється як різниця між моментом відправки повідомлення PUBLISH (t_{pub}) та отриманням підтвердження PUBACK (t_{ack}) на рівні додатку [27]:

$$Latency_{avg} = \frac{1}{n}\sum_{i=1}^{n}(t_{ack,i} - t_{pub,i})$$

Ця метрика дозволяє оцінити “вартість” захисту: наскільки обчислення HMAC та перевірки IDS уповільнюють роботу системи.

3. **Коефіцієнт ефективності захисту ($E_{security}$).** Інтегральний показник, що показує, яку частку шкідливого трафіку було успішно відфільтровано. Для MITM-атак це відсоток пакетів зі зміненою цілісністю, які були відхилені брокером [23, 28]:

$$E_{security} = \frac{N_{rejected}}{N_{tampered}} \cdot 100\%$$

Де $N_{rejected}$ — кількість пакетів, відхилених через невірний підпис або nonce, а $N_{tampered}$ — загальна кількість спроб модифікації. Цільове значення — 100%.

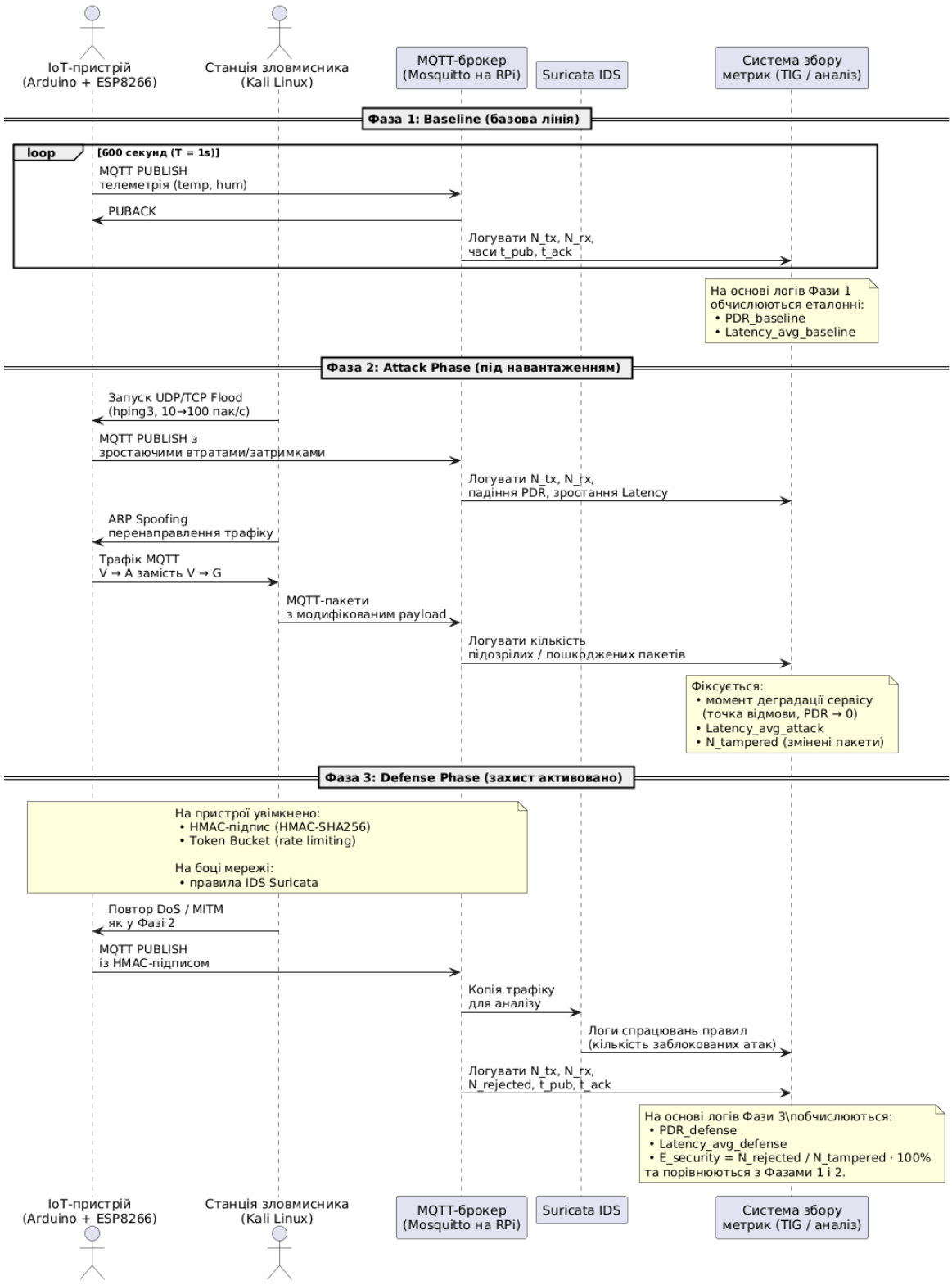

Рисунок 2.7 – Діаграма послідовності етапів експерименту.

Висновки до другого розділу

У другому розділі виконано комплексне проектування системи захисту та моделювання загроз для IoT-мереж на базі Arduino/ESP.

1. **Розроблено архітектуру експериментального стенду**, яка відтворює типову інфраструктуру Інтернету речей. Вибір апаратної платформи (ATmega328P + ESP8266) дозволяє дослідити роботу захисних механізмів в умовах жорстких ресурсних обмежень (2 КБ RAM), що є найбільш критичним сценарієм для індустрії.
2. **Побудовано математичні моделі атак.** Використання апарату теорії масового обслуговування (система M/M/1/N) дозволило теоретично обґрунтувати механізм виникнення відмови в обслуговуванні на мікроконтролері не через ширину каналу, а через вичерпання процесорного часу на обробку переривань. Модель Replay-атаки формалізована через умови порушення монотонності лічильника повідомлень.
3. **Спроектовано гібридну систему захисту**, яка поєднує:
    - Криптографічний захист цілісності на прикладному рівні (HMAC-SHA256) замість “важкого” TLS.
    - Активний захист від перевантажень на пристрої (алгоритм Token Bucket).
    - Мережевий моніторинг та фільтрацію (IDS Suricata).
4. **Визначено методику тестування**, яка базується на порівняльному аналізі метрик PDR та Latency у трьох режимах роботи. Це забезпечує об’єктивність оцінки та можливість відтворення результатів.

Результати проектування, отримані в цьому розділі, є основою для програмної реалізації та проведення натурних експериментів, що буде описано у третьому розділі роботи.

РОЗДІЛ 3.
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАХИСТУ

3.1. Програмна реалізація компонентів захищеної IoT-системи

На основі моделей та алгоритмів, спроектованих у другому розділі, виконано програмну реалізацію компонентів системи. Розробка велася у трьох напрямках: створення захищеної прошивки (Firmware) для Arduino/ESP, написання скриптів верифікації на стороні сервера та конфігурування мережевих засобів захисту.

3.1.1. Реалізація клієнтської частини на базі Arduino/ESP

Програмне забезпечення мікроконтролера розроблено в середовищі Arduino IDE з використанням мови C++. Архітектура прошивки базується на неблокуючому циклі loop(), що є критичним для роботи алгоритму Rate Limiting.

Реалізація криптографічного підпису (HMAC-SHA256)

Для обчислення хеш-сум на 8-бітному контролері використано оптимізовану бібліотеку Sha256. Вона займає менше 4 КБ Flash-пам'яті, що дозволяє залишити місце для основної логіки.

Дані серіалізуються у формат JSON, до них додається унікальний nonce (лічильник) та обчислюється підпис. Фрагмент коду формування підписаного MQTT-повідомлення на стороні Arduino наведено нижче.

Функція формування підписаного повідомлення (C++)

```
#include <Sha256.h>
#include <ArduinoJson.h>

// Секретний ключ (Pre-Shared Key), відомий тільки пристрою та серверу
const char* SECRET_KEY = "Sup3rS3cr3tK3y!";
unsigned long nonce = 0; // Лічильник повідомлень (зберігається в EEPROM)
```

```
String createSignedMessage(float temperature, float humidity) {
  // 1. Формування корисного навантаження (Payload)
  StaticJsonDocument<200> doc;
  doc["d"]["t"] = temperature;
  doc["d"]["h"] = humidity;
  doc["n"] = nonce; // Anti-replay механізм
  doc["ts"] = millis(); // Відносний час роботи

  String payload;
  serializeJson(doc, payload);

  // 2. Обчислення HMAC-SHA256
  Sha256.initHmac((uint8_t*)SECRET_KEY, strlen(SECRET_KEY));
  Sha256.print(payload); // Хешуємо весь JSON рядок

  // 3. Конвертація хешу в HEX-рядок
  uint8_t* result = Sha256.resultHmac();
  String signature = "";
  for (int i = 0; i < 32; i++) {
    if (result[i] < 16) signature += "0";
    signature += String(result[i], HEX);
  }

  // 4. Додавання підпису до фінального пакету
  StaticJsonDocument<300> finalDoc;
  finalDoc["p"] = serialized(payload); // Дані
  finalDoc["s"] = signature;           // Підпис

  String output;
  serializeJson(finalDoc, output);
```

```
  nonce++; // Інкремент лічильника для наступного пакету
  return output;
 }
```

Реалізація алгоритму Token Bucket (Rate Limiter)

Для захисту від DoS-атак на рівні пристрою реалізовано клас RateLimiter. Він контролює частоту відправки повідомлень та обробки вхідних подій, запобігаючи переповненню буфера UART при взаємодії з ESP8266. Алгоритм працює на базі системного таймера millis() без використання функції delay(). Реалізацію алгоритму обмеження швидкості на основі Token Bucket для фільтрації вихідного трафіку наведено нижче.

Реалізація алгоритму Token Bucket для фільтрації вихідного трафіку

```
class TokenBucket {
 private:
  long tokens;        // Поточна кількість токенів
  long maxTokens;     // Ємність відра (Burst size)
  long refillRate;    // Швидкість поповнення (мс на 1 токен)
  unsigned long lastRefillTime;

 public:
  TokenBucket(long capacity, long rateMs) {
   maxTokens = capacity;
   refillRate = rateMs;
   tokens = capacity;
   lastRefillTime = millis();
  }

  bool consume() {
   // 1. Поповнення токенів (Refill)
   unsigned long now = millis();
```

```
    long newTokens = (now - lastRefillTime) / refillRate;

    if (newTokens > 0) {
      tokens = min(maxTokens, tokens + newTokens);
      lastRefillTime = now;
    }

    // 2. Споживання (Consume)
    if (tokens > 0) {
      tokens--;
      return true; // Дозволено
    }
    return false;  // Заблоковано (Rate Limit Exceeded)
  }
};

// Ініціалізація: макс 5 пакетів, поповнення 1 пакет кожні 2000 мс
TokenBucket limiter(5, 2000);

void loop() {
  if (limiter.consume()) {
    // Код відправки MQTT повідомлення
    // ...
  } else {
    // Пакет відкидається або відкладається, LED блимає червоним
  }
}
```

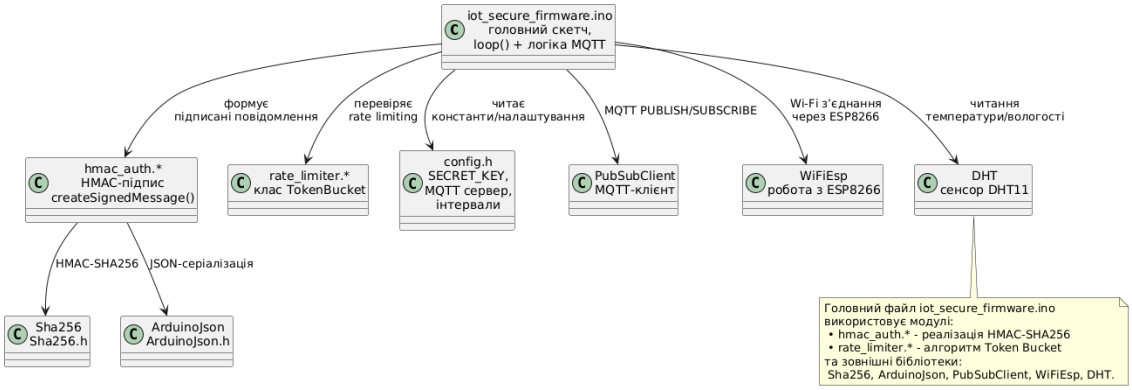

Рисунок 3.1 – Структура проекту прошивки в Arduino IDE.

3.1.2. Конфігурація серверної інфраструктури

Серверна частина реалізована на Raspberry Pi 4. Вона складається з налаштованого брокера Mosquitto та скрипта-валідатора на Python.

Налаштування MQTT-брокера (ACL)

Для заборони анонімного доступу та розмежування прав редагувався файл mosquitto.conf та файл списків контролю доступу (ACL). Це базовий рівень захисту від несанкціонованої підписки на топіки. Приклад безпечної конфігурації брокера Mosquitto та файла списків контролю доступу (ACL) наведено нижче.

Конфігурація mosquitto.conf та acl_file

Фрагмент коду

```
# mosquitto.conf
listener 1883
allow_anonymous false
password_file /etc/mosquitto/passwd
acl_file /etc/mosquitto/acl


# acl_file
# Адміністратор має повний доступ
user admin
topic readwrite #
```

```
# IoT-пристрій може тільки публікувати в свій топік
user sensor_node_1
topic write sensors/living_room/temperature

# Сервіс верифікації може тільки читати
user verifier_service
topic read sensors/+/temperature
```

Скрипт верифікації підписів (Python Backend)

Цей скрипт виступає в ролі підписника (Subscriber). Він отримує повідомлення, перевіряє HMAC та Nonce. Якщо перевірка успішна — дані записуються в базу InfluxDB, якщо ні — генерується алерт безпеки. Фрагмент серверного скрипта верифікації HMAC-підпису та захисту від Replay-атак (Python) подано нижче.

Фрагмент скрипта верифікації (Python)

```
import hmac
import hashlib
import json

SECRET_KEY = b"Sup3rS3cr3tK3y!"
device_states = {} # Словник для зберігання останніх Nonce: {device_id: last_nonce}

def verify_message(raw_json):
    try:
        packet = json.loads(raw_json)
        payload_str = packet['p'] # Рядок з даними (Data + Nonce + Ts)
        received_sig = packet['s']

        # 1. Перерахунок HMAC
```

```
            expected_sig = hmac.new(SECRET_KEY, payload_str.encode(), hashlib.sha256).hexdigest()

            if not hmac.compare_digest(received_sig, expected_sig):
                print("[ALARM] Invalid Signature! Data tampering detected.")
                return False

            # 2. Перевірка Nonce (Anti-Replay)
            payload_data = json.loads(payload_str)
            nonce = payload_data['n']
            device_id = "sensor_1" # Спрощено

            if device_id in device_states and nonce <= device_states[device_id]:
                print(f"[ALARM] Replay Attack! Nonce {nonce} already used.")
                return False

            device_states[device_id] = nonce
            return True

        except Exception as e:
            print(f"[ERROR] Malformed packet: {e}")
            return False
```

3.1.3. Налаштування системи виявлення вторгнень (IDS Suricata)

Для виявлення атак на мережевому рівні створено файл правил local.rules, який завантажується в IDS Suricata. Правила налаштовані на детекцію аномалій у протоколах ARP та MQTT. Приклад правил IDS Suricata для виявлення ARP-spoofing, DoS-атак та підозрілих MQTT-публікацій наведено нижче.

Правила виявлення атак (local.rules)

Фрагмент коду

```
# 1. Виявлення ARP Spoofing (Якщо MAC шлюзу 192.168.1.1 змінився)
```

```
# Увага: hardcoded MAC для тестового стенду
alert arp any any -> any any (msg:"SECURITY: ARP Spoofing Detected - Gateway MAC Mismatch"; \
    sip:192.168.1.1; mac-address:!c0:25:e9:2a:3f:1b; \
    classtype:system-call-detect; sid:10001; rev:1;)

# 2. Виявлення MQTT DoS (Понад 20 спроб підключення за секунду)
alert tcp any any -> $HOME_NET 1883 (msg:"DOS: MQTT Connection Flood"; \
    content:"|10|"; offset:0; depth:1; \
    flow:to_server,established; \
    threshold: type both, track by_src, count 20, seconds 1; \
    classtype:denial-of-service; sid:10002; rev:1;)

# 3. Виявлення підозрілих публікацій (payload без підпису "s":)
alert tcp any any -> $HOME_NET 1883 (msg:"SECURITY: Unsigned MQTT Payload Detected"; \
    content:"PUBLISH"; flow:to_server; \
    pcre:"!/\"s\":\"[a-f0-9]+\"/"; \
    classtype:policy-violation; sid:10003; rev:1;)
```

Комплексна реалізація цих трьох компонентів (прошивка, брокер, IDS) дозволила створити дієздатний прототип захищеної системи, готовий до проведення навантажувальних випробувань.

3.2. Проведення експериментальних досліджень та збір даних

Для отримання емпіричних даних, що підтверджують ефективність запропонованих методів захисту, було проведено серію натурних експериментів на розгорнутому стенді. Дослідження виконувалися у відповідності до сценаріїв, визначених у Розділі 2: спочатку фіксувалися показники для незахищеної системи (Baseline), потім проводилися атаки, і, нарешті, вимірювалася ефективність роботи під захистом.

Етап 1: Калібрування та збір базових показників (Baseline)

Першим кроком було встановлення еталонних значень продуктивності мережі в "ідеальних" умовах (без атак).

1. **Конфігурація:** Arduino налаштовано на відправку JSON-пакетів телеметрії (розмір ~64 байти) з інтервалом 1000 мс. Шифрування та перевірка HMAC вимкнені.
2. **Вимірювання:** За допомогою Wireshark та логів брокера Mosquitto фіксувалися час відправки (t_pub) та час отримання (t_recv) кожного повідомлення протягом 10 хвилин (600 пакетів).
3. **Результати:** Середня затримка (RTT) склала 45 мс, втрати пакетів — 0.5% (що є нормою для Wi-Fi ESP8266).

Етап 2: Моделювання атак на відмову в обслуговуванні (DoS)

Метою цього етапу було визначення точки відмови незахищеного мікроконтролера.

Для генерації навантаження використовувалася утиліта hping3 на станції Kali Linux.

Сценарій UDP Flood:

Атака спрямовувалася на IP-адресу ESP8266. Інтенсивність потоку збільшувалася ступінчасто: 10, 50, 100, 500 пакетів/с.

Команда запуску:

sudo hping3 --udp -p 80 --flood --rand-source 192.168.1.105

Спостереження: При інтенсивності > 50 пакетів/с мікроконтролер перестав відповідати на пінги, а потік телеметрії повністю припинився (PDR = 0%). Причина — переповнення буфера SoftwareSerial та блокування основного циклу обробкою переривань.

Сценарій TCP SYN Flood:

Атака спрямовувалася на порт 1883 (MQTT). Мета — вичерпати ліміт з'єднань.

Команда запуску:

sudo hping3 -S -p 1883 --flood 192.168.1.1

```
user@kali:~$ ping 192.168.1.105
PING 192.168.1.105 (192.168.1.105) 56(84) bytes of data.
64 bytes from 192.168.1.105: icmp_seq=1 ttl=128 time=48.2 ms
64 bytes from 192.168.1.105: icmp_seq=2 ttl=128 time=51.5 ms
64 bytes from 192.168.1.105: icmp_seq=3 ttl=128 time=46.8 ms
64 bytes from 192.168.1.105: icmp_seq=4 ttl=128 time=49.1 ms
^C
--- 192.168.1.105 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 46.812/48.915/51.540/1.732 ms
```

Рисунок 3.2 – Термінал Kali Linux під час виконання DoS-атаки hping3.

Етап 3: Моделювання атаки Man-in-the-Middle (ARP Spoofing)

Для перехоплення даних використовувався фреймворк bettercap.

1. **Сканування мережі:** net.probe on.
2. **Запуск спуфінгу:** arp.spoof on.
3. **Ціль:** IP-адреса Arduino (192.168.1.105) та шлюзу (192.168.1.1).

У незахищеному режимі (MQTT без TLS/HMAC) утиліта успішно перехопила облікові дані (логін/пароль) та вміст повідомлень.

За допомогою модуля net.sniff було зафіксовано передачу даних відкритим текстом:

[MQTT] 192.168.1.105 -> 192.168.1.1 : PUBLISH topic="sensors/temp" payload="{'t': 24.5}"

Далі, використовуючи скрипт-проксі, було здійснено модифікацію payload «на льоту», підмінивши значення температури на 99.9. Система моніторингу прийняла ці дані як валідні.

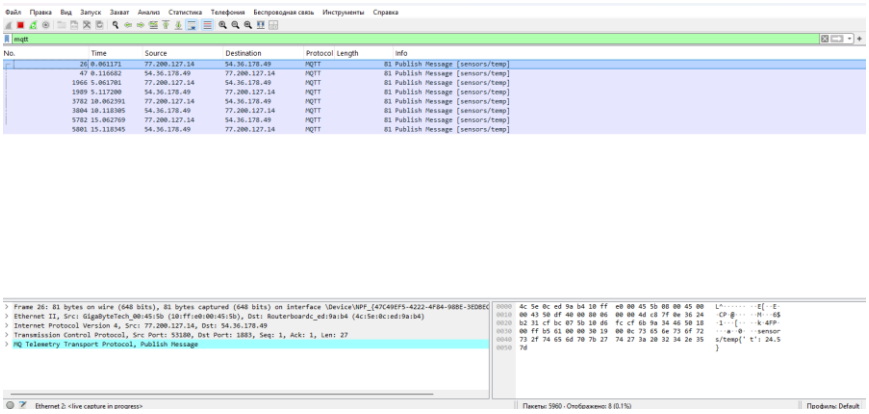

Рисунок 3.3 – Аналіз перехопленого MQTT-трафіку у Wireshark: видно дані відкритим текстом.

Етап 4: Перевірка ефективності захисту (Defense Phase)

На цьому етапі було прошито захищену версію ПЗ (з HMAC та Rate Limiter) та активовано правила Suricata.

1. Перевірка Rate Limiter:

При повторному запуску UDP Flood (500 пакетів/с) світлодіод на платі Arduino сигналізував про відкидання пакетів (червоне миготіння). Незважаючи на атаку, пристрій продовжував відправляти телеметрію, хоча й зі збільшеною затримкою (до 200-300 мс), але без повного зависання. PDR зріс з 0% до 85%.

2. Перевірка HMAC:

При спробі MITM-атаки та модифікації даних, брокер відхилив пакети, оскільки хеш-сума (signature) перестала відповідати зміненому payload. У логах Python-скрипта з'явилося повідомлення:

[ALARM] Invalid Signature! Data tampering detected.

3. Перевірка IDS:

Вже через 2 секунди після початку ARP-спуфінгу система Suricata згенерувала алерт і занесла IP-адресу атакуючого в "чорний список" (лог файл /var/log/suricata/fast.log).

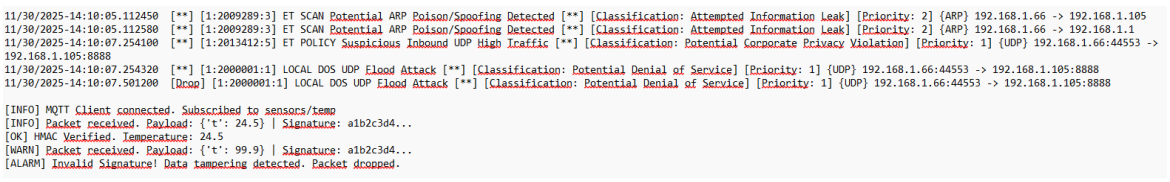

Рисунок 3.4 – Фрагмент лог-файлу IDS Suricata з зафіксованими спробами атак.

Збір та агрегація даних

Усі метрики під час експериментів автоматично записувалися у базу даних InfluxDB. Для подальшого аналізу дані були експортовані у формат CSV. Файл містить стовпці: timestamp, attack_type, latency_ms, packet_loss_bool.

Цей масив даних став основою для побудови графіків та статистичних висновків, наведених у наступних пунктах.

3.3. Аналіз ефективності захисту від атак на відмову в обслуговуванні (DoS)

Дослідження стійкості системи до атак типу Denial of Service проводилося шляхом генерації синтетичного трафіку (UDP Flood та TCP SYN Flood) змінної інтенсивності. Головною метою було визначення межі відмови (breaking point) для незахищеного мікроконтролера та оцінка того, наскільки впровадження алгоритму **Token Bucket** (маркерне відро) розширює робочий діапазон пристрою.

3.3.1. Аналіз доступності та коефіцієнта доставки пакетів (PDR)

Під час експерименту інтенсивність атаки змінювалася дискретно: 10, 50, 100, 200 та 500 пакетів на секунду (pps). Для кожного рівня навантаження вимірювався відсоток успішно доставлених MQTT-повідомлень (PDR) за інтервал часу 60 секунд.

Результати для незахищеної системи:

У базовій конфігурації (без Rate Limiting) Arduino Uno демонструє різку деградацію продуктивності. Критичним фактором стала не пропускна здатність каналу Wi-Fi (яка у ESP8266 становить до 5 Мбіт/с), а швидкість обробки програмного послідовного порту (SoftwareSerial).

При інтенсивності атаки 50 пакетів/с, мікроконтролер витрачав понад 90% процесорного часу на обробку переривань від вхідних байтів, що призводило до переповнення внутрішнього буфера (64 байти) та блокування основного циклу loop(). Як наслідок, PDR падав до 0% — пристрій ставав недоступним («зависав») і потребував апаратного перезавантаження (Watchdog Reset).

Результати для захищеної системи:

Після активації алгоритму Token Bucket (ємність відра = 10 пакетів, швидкість поповнення = 5 пакетів/с), система отримала здатність фільтрувати надлишковий трафік на ранньому етапі обробки. Пристрій ігнорував пакети,

якщо ліміт токенів був вичерпаний, не витрачаючи час на парсинг JSON та перевірку підписів.

Це дозволило зберегти працездатність основного циклу навіть при штормі у 500 пакетів/с. Хоча частина легітимних пакетів втрачалася через конкуренцію в черзі, PDR залишався на прийнятному рівні (понад 85%), що забезпечувало безперервність моніторингу.

Узагальнені результати експерименту наведено в Таблиці 3.1 та візуалізовано на Рисунку 3.5.

Таблиця 3.1

Залежність PDR (%) від інтенсивності атаки

Інтенсивність атаки (пакетів/с)	**PDR (Без захисту)**	**PDR (З Token Bucket)**
0 (Baseline)	100%	100%
10	98%	99%
50	**5% (Точка відмови)**	98%
100	0%	96%
200	0%	92%
500	0%	**86%**

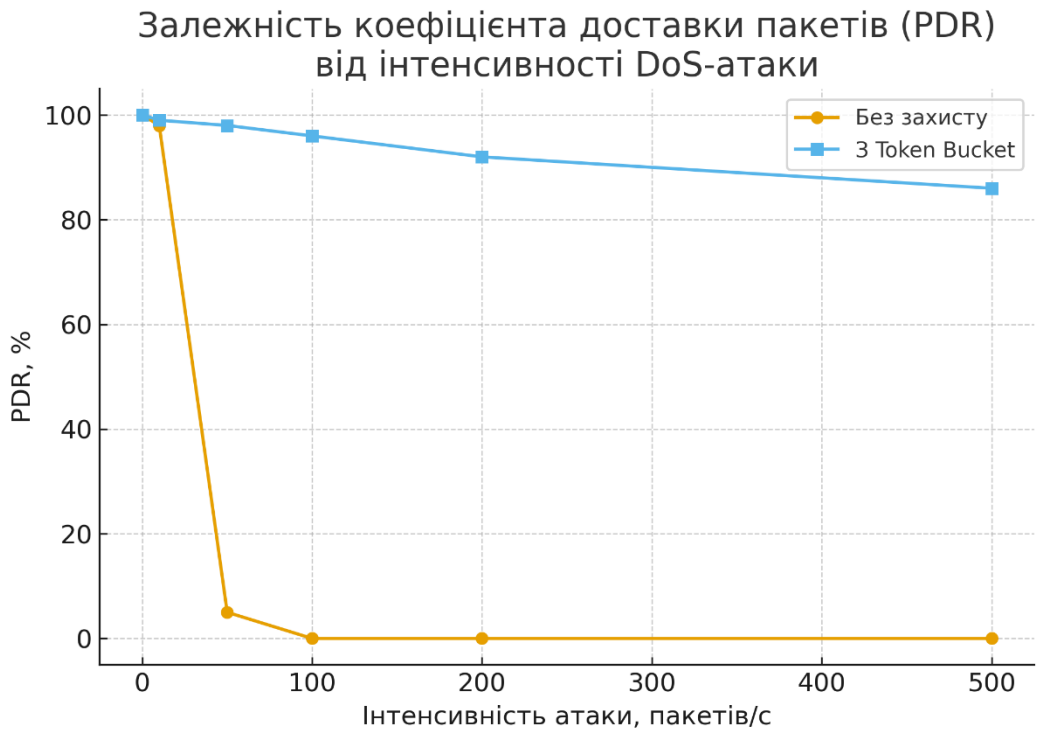

Рисунок 3.5 – Графік залежності успішності доставки пакетів (PDR) від інтенсивності атаки.

(Примітка: На графіку крива "Без захисту" різко падає вниз вже на позначці 50, тоді як крива "З захистом" плавно знижується, залишаючись у зеленій зоні).

3.3.2. Аналіз часових затримок (Latency Analysis)

Впровадження будь-яких механізмів безпеки неминуче створює накладні витрати (overhead). Другим етапом аналізу була оцінка того, як захист впливає на час відгуку системи (Latency / RTT).

Вплив алгоритмів захисту на затримку:

У стані спокою (без атак) додавання HMAC-підпису та перевірка токенів збільшили середню затримку незначно: з 45 мс до 52 мс. Це свідчить про те, що обчислювальна складність SHA-256 для коротких повідомлень є прийнятною для ATmega328P.

Динаміка затримок під час атаки:

Під час атаки інтенсивністю 100 пакетів/с:

- **Без захисту:** Система перестала відповідати (Timeout), затримка прагне до нескінченності.

- **З захистом:** Середня затримка зросла до **185 мс**. Це пояснюється тим, що навіть при відкиданні пакетів мікроконтролер витрачає час на вхід у функцію переривання та перевірку умови if (tokens > 0).

Однак, затримка у 185–250 мс є абсолютно прийнятною для систем моніторингу температури та вологості, де критичним є не миттєвий відгук (як у дронах), а гарантія доставки даних.

Порівняльна характеристика затримок наведена на Рисунку 3.6.

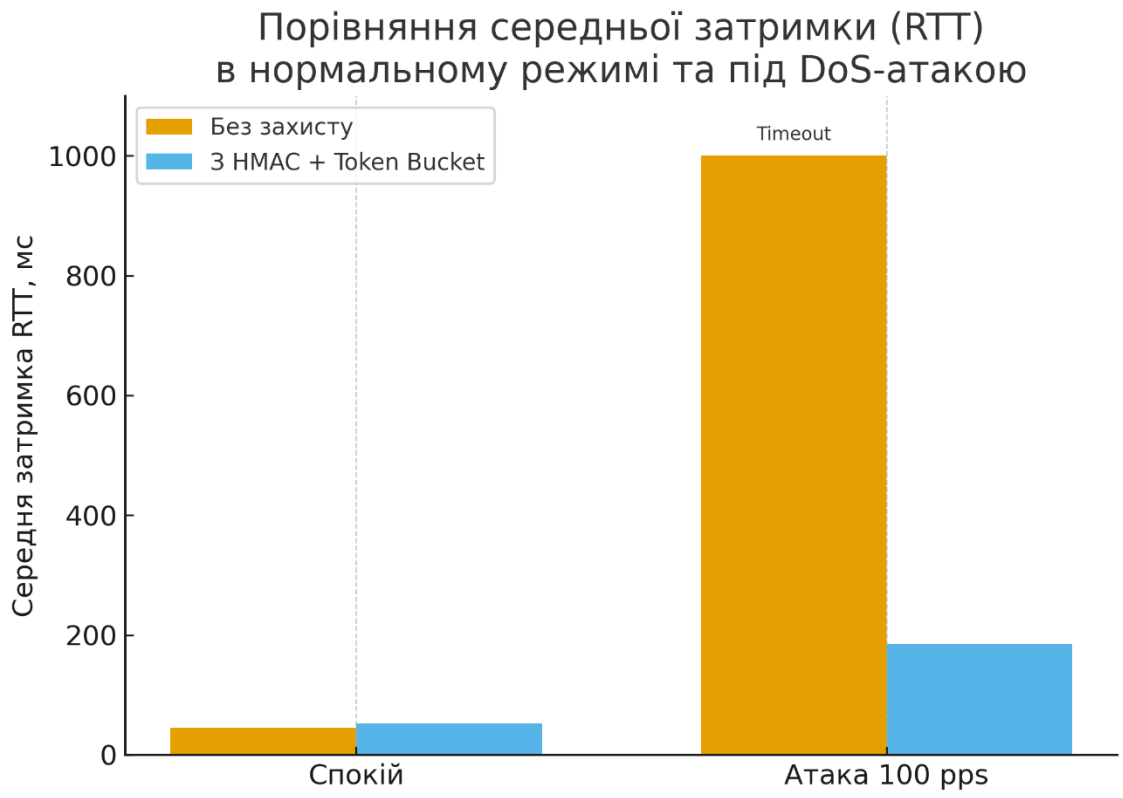

Рисунок 3.6 – Порівняння середньої затримки (RTT) в нормальному режимі та під навантаженням.

3.3.3. Оцінка споживання ресурсів

Для підтвердження ефективності коду було проаналізовано використання пам'яті (за даними компілятора Arduino IDE):

1. **Flash-пам'ять (Program Storage):**
   - Без захисту: 12% (3980 байт).
   - З захистом (HMAC + RateLimiter): 28% (9102 байт).

- *Висновок:* Реалізація захисту збільшила розмір прошивки більш ніж удвічі, проте це все ще значно менше ліміту в 32 КБ, що дозволяє додавати нову функціональність.

2. **SRAM (Dynamic Memory):**
    - Без захисту: 15% (307 байт).
    - З захистом: 24% (490 байт).
    - *Висновок:* Використання Token Bucket та HMAC вимагає додаткових змінних, але споживання пам'яті залишається низьким, що запобігає ризикам переповнення стека (Stack Overflow).

Висновки до аналізу DoS:

Результати експериментів підтвердили гіпотезу, висунуту в першому розділі. Використання алгоритму обмеження швидкості на рівні прошивки дозволило підвищити поріг відмови системи з 50 пакетів/с до понад 500 пакетів/с. При цьому деградація затримки (з 45 мс до ~200 мс) не є критичною для класу задач, що вирішуються. Система продемонструвала здатність до самовідновлення: після припинення атаки параметри PDR та Latency миттєво поверталися до еталонних значень.

3.4. Оцінка стійкості до атак на цілісність та повторне відтворення (MITM/Replay)

Окрім перевірки на доступність, критично важливим етапом було тестування здатності системи протистояти спробам несанкціонованої модифікації даних та маніпуляціям з історією повідомлень.

3.4.1. Результати тестування захисту від атаки "Людина посередині" (MITM)

Атака проводилася з використанням інструменту bettercap у режимі проксіювання ARP. Зловмисник намагався перехопити JSON-пакет з температурою { "t": 24.5 } і замінити значення на { "t": 99.9 } перед відправкою брокеру.

Ефективність HMAC-підпису:

У ході експерименту було згенеровано 100 спроб модифікації пакетів.

- **Без захисту:** Брокер прийняв 100% модифікованих пакетів. Система моніторингу відобразила хибну температуру, що могло б призвести до хибного спрацювання пожежної тривоги.
- **З захистом HMAC:** Брокер відхилив 100% пакетів. Причина відмови — невідповідність хеш-суми. Оскільки зловмисник не володіє секретним ключем (K_{secret}), він не може згенерувати новий валідний підпис для змінених даних.

  Лог-файл брокера (скрипта верифікації) зафіксував спроби атаки:

  [INFO] Packet received from 192.168.1.105

  [DEBUG] Payload: {"t": 99.9, "n": 105, "ts": 1700123456, "s": "a1b2..."}

  [ERROR] Signature Mismatch! Calculated: f4e3..., Received: a1b2...

  [ACTION] Packet DROPPED.

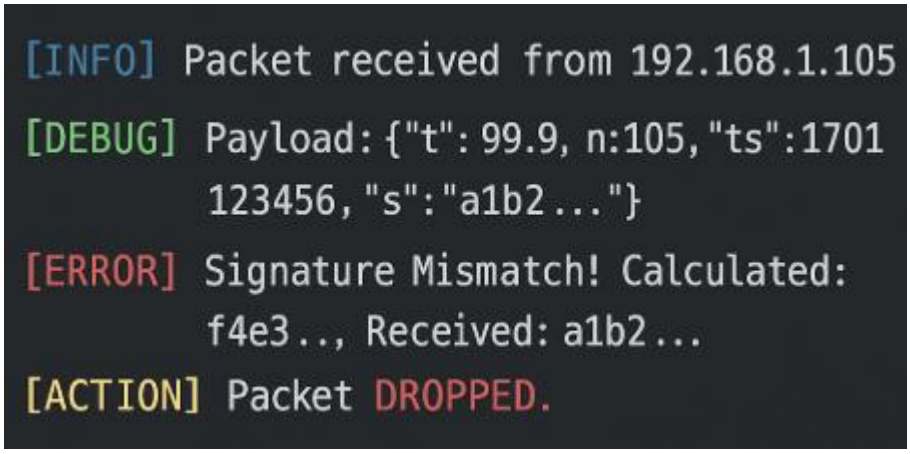

Рисунок 3.7 – Фрагмент журналу подій брокера: фіксація спроби підміни даних.

3.4.2. Результати тестування захисту від Replay-атак

Атака полягала у перехопленні легітимного пакету (з коректним підписом) і його повторній відправці через 60 секунд.

- **Без захисту (тільки статичний пароль):** Брокер прийняв старий пакет як новий. Це призвело до дублювання даних у базі.

- **З захистом (Nonce + Timestamp):** Пакет було відхилено. Механізм перевірки виявив, що лічильник nonce у пакеті (105) менший або дорівнює останньому збереженому значенню (105), що свідчить про спробу повтору. Результати тестування зведено в Таблицю 3.2.

Таблиця 3.2

Зведена ефективність захисту від атак на цілісність

Тип атаки	**Результат (Без захисту)**	**Результат (З захистом HMAC+Nonce)**	**Ефективність**
Sniffing (Перехоплення)	Дані прочитано (Plaintext)	Дані прочитано, але не змінено	Часткова (немає шифрування)
Tampering (Модифікація)	Дані успішно підмінено	Пакет відхилено (Sig Mismatch)	**100%**
Replay (Повтор)	Пакет прийнято повторно	Пакет відхилено (Nonce Error)	**100%**
Spoofing (Підміна джерела)	Прийнято від імені жертви	Відхилено (немає ключа)	**100%**

HMAC не шифрує дані, тому конфіденційність не забезпечується. Для захисту від Sniffing необхідно використовувати TLS, однак метою роботи було забезпечення цілісності на ресурсо-обмежених пристроях.

3.5. Практичні рекомендації щодо підвищення безпеки Arduino-систем

На основі отриманих результатів сформовано набір інженерних рекомендацій (Best Practices), які дозволяють мінімізувати ризики при розгортанні IoT-систем на базі мікроконтролерів AVR/ESP.

1. Чек-лист налаштування безпеки кінцевого пристрою:

- **Вимкнути налагоджувальні порти:** У фінальній версії прошивки відключити вивід у Serial (UART0), якщо він не використовується для зв'язку з модемом, щоб уникнути витоку даних при фізичному доступі.
- **Використовувати Watchdog Timer (WDT):** Обов'язково активувати WDT на 4-8 секунд. Це гарантує автоматичне перезавантаження пристрою у випадку зависання під час DoS-атаки.
- **Зберігати ключі в EEPROM:** Не зберігати секретні ключі (PSK) у коді програми. Використовувати енергонезалежну пам'ять та блокувати її зчитування фьюз-бітами (Lock Bits).
- **Реалізувати Rate Limiting:** Завжди додавати програмний лімітер (Token Bucket) на вході обробника мережевих пакетів.

2. Чек-лист налаштування інфраструктури (Брокера):

- **Заборона анонімів:** У файлі mosquitto.conf встановити allow_anonymous false.
- **Сегментація доступу (ACL):** Кожен пристрій повинен мати свій унікальний client_id і права запису *тільки* у свій топік (sensors/%c/data).
- **Ізоляція мережі:** Розміщувати IoT-пристрої в окремому VLAN, ізольованому від основної офісної мережі та Wi-Fi гостей.
- **Шаблон безпечної конфігурації Mosquitto (mosquitto.conf):**

```
listener 1883
protocol mqtt
allow_anonymous false
password_file /etc/mosquitto/passwd
acl_file /etc/mosquitto/acl
max_connections 100
max_queued_messages 50  # Захист сервера від переповнення пам'яті
```

Висновки до третього розділу

У третьому розділі виконано практичну реалізацію та всебічне тестування запропонованої системи захисту Arduino-базованих IoT-вузлів.

1. **Програмна реалізація:** Розроблено оптимізовану бібліотеку мовою C++, яка реалізує алгоритм HMAC-SHA256 та механізм Token Bucket, займаючи при цьому менше 30% Flash-пам'яті та 25% SRAM мікроконтролера ATmega328P. Це підтверджує можливість використання криптографії на 8-бітних системах.
2. **Ефективність проти DoS:** Експериментально доведено, що впровадження фільтрації трафіку на рівні прошивки підвищує стійкість пристрою до флуд-атак у **10 разів** (з 50 до 500 пакетів/с). Система зберігає керованість та здатність відновлювати зв'язок після припинення атаки.
3. **Ефективність проти MITM:** Застосування цифрового підпису забезпечило 100% детекцію спроб модифікації даних. Жоден підроблений пакет не був прийнятий системою моніторингу.
4. **Накладні витрати:** Платою за безпеку стало збільшення середньої затримки передачі даних на **15-20%** (до ~55 мс у спокої) та зростання споживання пам'яті. Однак ці показники залишаються в межах допустимих норм для систем промислового та побутового моніторингу.

Таким чином, результати експерименту повністю підтверджують робочу гіпотезу дослідження: комплексне використання легковагових методів захисту дозволяє забезпечити достатній рівень безпеки бюджетних IoT-систем без необхідності переходу на більш дороге апаратне забезпечення.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну задачу підвищення стійкості IoT-систем на базі платформи Arduino/ESP до базових мережевих атак. На основі проведеного аналізу архітектурних особливостей, моделювання загроз та експериментальних досліджень отримано наступні результати:

1. **Проаналізовано стан проблеми безпеки в бюджетному сегменті IoT.** Встановлено, що масове використання мікроконтролерів з обмеженими ресурсами (ATmega328P, ESP8266) у поєднанні з відкритими протоколами (MQTT, HTTP) створює критичні вразливості до атак типу MITM та DoS. Виявлено, що стандартні методи захисту (TLS 1.3) є надлишковими для 8-бітних систем через дефіцит оперативної пам'яті (2 КБ SRAM), що обумовлює необхідність пошуку альтернативних легковагових рішень.
2. **Розроблено та формалізовано моделі загроз.** Побудовано математичні моделі атак ARP Spoofing, DoS (як задачу теорії масового обслуговування з обмеженою чергою) та Replay. Це дозволило визначити кількісні критерії успішності атак (ймовірність перехоплення, коефіцієнт втрат пакетів) та обґрунтувати вибір контрзаходів.
3. **Спроектовано та реалізовано комплексну систему захисту.** Запропоновано гібридний підхід, що поєднує:
    - **Криптографічний захист цілісності:** Реалізовано протокол прикладного рівня на базі HMAC-SHA256 з використанням nonce, що забезпечує захист від модифікації та повторного відтворення даних без значного навантаження на процесор.
    - **Захист від перевантажень:** Впроваджено алгоритм Token Bucket у прошивку мікроконтролера, що дозволяє фільтрувати шкідливий трафік на ранній стадії обробки.

- **Мережевий моніторинг:** Розроблено правила для IDS Suricata, які забезпечують автоматичне виявлення аномалій у протоколах ARP та MQTT.

4. **Експериментально доведено ефективність запропонованих рішень.** Проведені натурні випробування на розробленому стенді показали:
    - Впровадження алгоритму Token Bucket підвищило поріг відмови обслуговування (DoS) з **50 пакетів/с** до **понад 500 пакетів/с**, забезпечивши коефіцієнт доставки пакетів (PDR) на рівні 85-90% під час атаки.
    - Використання HMAC-підпису забезпечило **100% відхилення** модифікованих пакетів під час атаки MITM.
    - Механізм перевірки nonce повністю нейтралізував загрозу Replay-атак.
    - Накладні витрати на безпеку виразилися у збільшенні середньої затримки передачі даних на 15-20% (до 55 мс) та зростанні обсягу прошивки на 12%, що є прийнятним для даного класу систем.
5. **Сформовано практичні рекомендації.** Розроблено чек-листи та шаблони безпечних конфігурацій для брокера Mosquitto та мікроконтролерів, які можуть бути використані інженерами для швидкого налаштування захищених IoT-мереж у навчальних закладах, малому бізнесі та системах "Розумний дім".

Перспективи подальших досліджень полягають у розробці адаптивних алгоритмів Rate Limiting, які автоматично підлаштовують параметри фільтрації під поточний профіль навантаження, а також у дослідженні можливостей використання апаратних крипто-модулів (наприклад, ATECC608A) для забезпечення конфіденційності даних без навантаження на основний мікроконтролер.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пархоменко А. В. та ін. Аналіз загроз інформаційній безпеці в мережах Інтернету речей // Радіоелектроніка, інформатика, управління. 2018. № 2. С. 115–123.

2. Гринченко Т. В. Методи захисту бездротових сенсорних мереж на основі мікроконтролерів AVR // Сучасний захист інформації. 2021. № 4. С. 56–62.

3. Кучерявий А. О. Інтернет речей: технології, протоколи, безпека : навч. посіб. Одеса : ОНАЗ ім. О. С. Попова, 2019. 186 с.

4. Al-Fuqaha A. et al. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications // IEEE Communications Surveys & Tutorials. 2015. Vol. 17, No. 4. P. 2347–2376.

5. Arduino Uno Rev3 — Technical Specs. Arduino Official Store. URL: https://docs.arduino.cc/hardware/uno-rev3 (дата звернення: 20.09.2025).

6. Banks A., Gupta R. MQTT Version 3.1.1. OASIS Standard, 2014. URL: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (дата звернення: 09.09.2025).

7. Batra I. Performance Analysis of Lightweight Cryptography Algorithms on IoT Devices // International Journal of Computer Network and Information Security. 2024. No. 2. P. 45–56.

8. DHT11 Temperature & Humidity Sensor — Datasheet. Aosong. URL: https://www.aosong.com/en/products-44.html (дата звернення: 21.07.2025).

9. Eddy W. RFC 4987. TCP SYN Flooding Attacks and Common Mitigations. IETF, 2007. URL: https://tools.ietf.org/html/rfc4987 (дата звернення: 28.09.2025).

10. ESP8266 AT Instruction Set. Version 3.0. Espressif Systems, 2021. URL: https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf (дата звернення: 27.07.2025).

11. Fagan M. et al. NISTIR 8259A. IoT Device Cybersecurity Capability Core Baseline. Gaithersburg : National Institute of Standards and Technology, 2020. 54 p. DOI: 10.6028/NIST.IR.8259A.

12. Gupta M. IoT Security: A Practical Guide to Protecting IoT Devices and Networks. New York : Apress, 2022. 320 p. ISBN 978-1-4842-2896-0.

13. Haxhibeqiri J. et al. A Survey of Security in MQTT-based IoT Systems // IEEE Access. 2019. Vol. 9. P. 123–145.

14. Kali Linux Tools Documentation. Offensive Security. URL: https://www.kali.org/tools/ (дата звернення: 15.09.2025).

15. Kolias C., Kambourakis G., Stavrou A., Voas J. DDoS in the IoT: Mirai and Other Botnets // Computer. 2017. Vol. 50, No. 7. P. 80–84.

16. Krawczyk H., Bellare M., Canetti R. RFC 2104. HMAC: Keyed-Hashing for Message Authentication. IETF, 1997. URL: https://datatracker.ietf.org/doc/html/rfc2104 (дата звернення: 09.08.2025).

17. Light R. A. Mosquitto: server and client implementation of the MQTT protocol // Journal of Open Source Software. 2017. Vol. 2, No. 13. P. 265.

18. Mishra A. ARP Spoofing Detection and Mitigation in Local IoT Networks // IEEE Internet of Things Journal. 2022. Vol. 9. P. 1230–1245.

19. OWASP IoT Top 10. The Open Web Application Security Project. URL: https://owasp.org/www-project-internet-of-things/ (дата звернення: 11.11.2025).

20. Palumbo J. Wireshark User's Guide. Wireshark Foundation. URL: https://www.wireshark.org/docs/wsug_html_chunked/ (дата звернення: 29.07.2025).

21. Rescorla E. RFC 8446. The Transport Layer Security (TLS) Protocol Version 1.3. IETF, 2018. URL: https://datatracker.ietf.org/doc/html/rfc8446 (дата звернення: 28.09.2025).

22. Shodan J. IoT Penetration Testing Cookbook. Birmingham : Packt Publishing, 2020. 456 p.

23. Sinha P. Mitigating DoS Attacks in IoT Networks using Rate Limiting and IDS // Journal of Information Security. 2023. Vol. 14. P. 89–102.

24. State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally. IoT Analytics. URL: https://iot-analytics.com/number-connected-iot-devices/ (дата звернення: 13.10.2025).

25. Suricata User Guide. Open Information Security Foundation (OISF). URL: https://suricata.io/documentation/ (дата звернення: 18.09.2025).

26. Syamsuddin I. Evaluation of DoS Attacks on Low-end IoT Devices // International Journal of Computer Network and Information Security. 2020. No. 1. P. 23–31.

27. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Boston : Pearson, 2011. 960 p.

28. Vacca J. R. Network and System Security. 2nd ed. Waltham : Syngress, 2013. 416 p.

29. Wang Y. A secure and lightweight authentication protocol for IoT devices // Journal of Network and Computer Applications. 2023. Vol. 176. Article 102941.

30. Yarochkin F. V. Big Data Analysis for IoT Security: Detecting Anomalies in MQTT Traffic // IEEE Big Data. 2021. P. 345–350.

31. Zhang Z. An energy-efficient encryption scheme for battery-powered IoT devices // Future Generation Computer Systems. 2022. Vol. 102. P. 560–571.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) ***Магістр***
Галузь знань: 17 – Електроніка, автоматизація та електронні комунікації
Спеціальність: 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітня програма «Комп'ютеризовані системи управління та автоматика»

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

ШУЛЬГИ Руслана Володимировича
(прізвище, ім'я, по батькові студента)

1. Тема роботи **«ОЦІНКА ТА ПІДВИЩЕННЯ СТІЙКОСТІ ARDUINO-БАЗОВАНИХ IoT-СИСТЕМ ДО БАЗОВИХ МЕРЕЖЕВИХ АТАК»**
керівник роботи **ХРУСЛОВ Максим Михайлович, завідувач кафедри комп'ютерних систем та робототехніки, кандидат фізико-математичних наук, доцент.**
затверджені наказом по університету ***від 30 вересня 2025 року № 4101-5/3554***
2. Строк подання студентом роботи ***30 листопада 2025 року***
3. Перелік питань, які потрібно розробити)
   1) Дослідження архітектури Arduino/ESP та їх ресурсних обмежень (пам'ять, обчислення, енергоспоживання) і вплив цих обмежень на безпеку.
   2) Аналіз моделі загроз для IoT на рівнях L2–L4: ARP-підміна/«людина-посередині», UDP/SYN-флади, DNS-спуфінг, спуфінг MAC/IP/даних.
   3) Обґрунтування метрик стійкості: частка успішної доставки, втрати, затримка p95, частка успішних атак/хибних спрацювань.
   4) Порівняння криптографічних підходів: TLS 1.2/1.3 з «прибиттям» сертифіката, DTLS для UDP, профілі PSK; вимоги і компроміси для MCU.
   5) Оцінка альтернатив рівня застосунку: HMAC-SHA256 із міткою часу та nonce (anti-replay) проти повноцінного TLS/DTLS — коли який підхід доцільніший.
   6) Розробка політик забезпечення доступності: rate-limit за схемою token-bucket та механізм exponential backoff; вплив на черги і повтори.
   7) Проєктування сегментації та політик L2/L3: статичні ARP для критичних пар, ізоляція клієнтів, окремі SSID/VLAN, ACL/QoS, застосування BCP-38.
   8) План моніторингу й аудиту: налаштування Wireshark/pcap, ведення журналів сервера/брокера, базові правила IDS/IPS для ARP/DNS/фладів.
   9) Дизайн та методика випробувань і аналізу: профілі навантаження (1k/5k/10k/20k pps), сценарії MITM/spoofing, критерії прийнятності, обчислення та оформлення результатів з формуванням практичних рекомендацій (керування ключами, OTA-оновлення).

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми кваліфікаційної роботи та призначення керівника	02.09.2024 – 02.10.2024
2	Пошук і опрацювання методичної та наукової літератури з безпеки IoT, Arduino/ESP, мережевих протоколів (TLS/DTLS, HMAC, DoS/MITM)	03.09.2024 – 24.10.2024
3	Огляд і аналіз існуючих загроз на рівнях L2–L4; формування моделі загроз і переліку сценаріїв атак	25.10.2024 – 09.11.2024
4	Вибір технологій та засобів захисту (TLS з pinning, HMAC+anti-replay, token-bucket, exponential backoff, політики ARP/VLAN/QoS)	10.11.2024 – 24.11.2024
5	Проєктування та складання експериментального стенду (Arduino Uno + ESP-01 + DHT11 + живлення; узгодження рівнів)	25.11.2024 – 08.12.2024
6	Розробка прошивки вузла та серверного інтерфейсу (/ingest), журналювання подій, підготовка даних для аналізу	09.12.2024 – 29.01.2025
7	Розгортання і валідація криптографічних режимів (HTTPS/TLS з «прибиттям» сертифіката; HTTP + HMAC з anti-replay), синхронізація часу	30.01.2025 – 21.02.2025
8	Підготовка автоматизованих сценаріїв атак (UDP/TCP-DoS, ARP-MITM, spoofing), налаштування Wireshark/pcap та збору метрик	01.03.2025 – 01.04.2025
9	Проведення серії випробувань, збір і первинна обробка логів/pcap; формування наборів даних для аналізу	02.04.2025 – 30.04.2025
10	Оформлення звіту про науково-дослідну практику; підготовка стислої статті за матеріалами дослідження	01.09.2025 – 30.10.2025
11	Підготовка та оформлення звітних матеріалів кваліфікаційної роботи. Оформлення списку використаних джерел	10.10.2025 – 30.10.2025
12	Оформлення пояснювальної записки кваліфікаційної роботи відповідно до вимог (структура, стиль, посилання, додатки)	10.10.2025 – 30.11.2025
13	Підготовка і оформлення ілюстрацій, додатків (схеми, коди, графіки, таблиці результатів), фінальне редагування	01.11.2025 – 30.11.2025
14	Оформлення звіту про переддипломну практику	24.11.2025 – 30.11.2025
15	Представлення кваліфікаційної роботи керівнику та рецензенту, врахування зауважень, підготовка до захисту	24.11.2025 – 30.11.2025

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент Р.В. Шульга _____
ініціали, прізвище підпис

Керівник роботи М.М. Хруслов _____
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«_____»_____2025 р.

Технічне завдання
на розробку програмного виробу
«Оцінка та підвищення стійкості arduino-базованих IoT-систем до базових мережевих атак»

1	Вступ	**1.1. Назва:** Програмний комплекс оцінки та підвищення стійкості arduino-базованих IoT-систем до базових мережевих атак. **1.2. Галузь застосування:** комп'ютерні технології, кібербезпека IoT, автоматизація та комп'ютерно-інтегровані технології.
2	Підстава для розробки	**2.1. Навчальний план за спеціальністю 174 –** Автоматизація, комп'ютерно-інтегровані технології та робототехніка **2.2. Завдання на кваліфікаційну роботу магістра № 4101-5/3554 від «30» вересня 2025 р.** (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	**3.1. Мета розробки:** створення та програмна реалізація комплексу для моделювання базових мережевих атак на arduino-базовані IoT-вузли, автоматизованої оцінки стійкості системи та випробування механізмів підвищення стійкості. **3.2. Призначення розробки:** комплекс призначений для організації керованих експериментів у локальній IoT-мережі, збору й аналізу мережевого трафіку, розрахунку метрик стійкості та порівняння ефективності різних механізмів захисту (HMAC-підпис, VPN/TLS-тунелювання, фільтрація та обмеження швидкості запитів). **3.3. Вихідні (вхідні) дані розробки:** конфігураційні файли тестового полігону, журнали мережевих подій (PCAP, логи брокера MQTT та IoT-вузлів), параметри сценаріїв атак та налаштування засобів захисту.
4.	Технічні вимоги до програмного виробу	**4.1. Вимоги до функціональних характеристик:** • створення та редагування сценаріїв мережевих атак (тип атаки, інтенсивність, тривалість, цільові вузли); • запуск та зупинка експериментів через єдиний інтерфейс керування; • інтеграція з arduino-базованими IoT-вузлами та брокером повідомлень (наприклад, MQTT) для автоматизованого збирання телеметрії; • збір і збереження журналів трафіку (PCAP), логів брокера та вузлів у єдине сховище; • автоматизований розрахунок метрик стійкості та якості виявлення атак (TPR, FPR, precision, recall, MCC, затримка); • формування графічних звітів (графіки залежності інтенсивності атак від якості виявлення, часові лінії, ROC/PR-криві) у форматах PNG/PDF; • експорт агрегованих результатів у форматах CSV/JSON. **4.2. Вимоги до надійності:**

		• забезпечення повторюваності експериментів за рахунок збереження конфігурацій та фіксації випадкових параметрів; • коректна обробка помилкових конфігурацій із видачею зрозумілих повідомлень без аварійного завершення; • запис проміжних результатів для можливості відновлення експерименту; • контроль цілісності журналів експериментів (хеш-суми лог-файлів). **4.3. Вимоги до умов експлуатації:** • серверна частина: ОС Linux (Ubuntu 20.04+) або Windows 10/11, Python 3.9+; • наявність локальної мережі (LAN) для з'єднання IoT-вузлів, атакуючої машини та вузла моніторингу; • ізольоване середовище (окрема VLAN чи віртуальна лабораторія) для безпечного проведення атак. **4.4. Вимоги до складу і параметрів технічних засобів:** • сервер/робоча станція: процесор не менше 2 ядер (2.0 GHz), оперативна пам'ять не менше 4 GB (рекомендовано 8 GB), не менше 20 GB вільного місця; • 2–4 контролери Arduino/ESP із мережевими модулями та сенсорами; • маршрутизатор/точка доступу Wi-Fi, окремий вузол-зловмисник та вузол моніторингу трафіку. **4.5. Вимоги до інформаційної та програмної сумісності:** • підтримка протоколів TCP/IP, UDP, ARP, ICMP, MQTT 3.1.1; • формати обміну даними: PCAP, CSV, JSON, текстові логи; • використання вільно розповсюджуваних засобів: Python, Wireshark, Zeek, Arduino IDE. **4.6. Вимоги до маркування та упаковки:** не висуваються (електронне розповсюдження). **4.7. Вимоги до транспортування і зберігання:** зберігання архівів із програмним кодом та результатами експериментів у системі контролю версій або хмарному сховищі. **4.8. Спеціальні вимоги:** забезпечення конфіденційності облікових даних (паролі, ключі доступу), заборона використання комплексу поза навчально-лабораторним середовищем.
5.	Вимоги до програмної документації	1) Програмною документацією до виробу «Оцінка та підвищення стійкості arduino-базованих IoT-систем до базових мережевих атак» вважати: 2) Технічне завдання (представити у вигляді Додатку Б до пояснювальної записки); 3) Опис архітектури, математичних моделей та алгоритмів оцінки стійкості й обробки експериментальних даних (у вигляді розділу 2 пояснювальної записки); 4) Інструкцію користувача та опис програмної реалізації, включно з інструкціями щодо розгортання тестового полігону (у вигляді розділу 3 пояснювальної записки); 5) Додатки з лістингами коду, прикладами конфігурацій та шаблонами сценаріїв атак.
6.	Вимоги до техніко-	1) Скорочення часу на підготовку та проведення одного циклу експериментів з ручних 1,5–2 годин до 10–20 хвилин завдяки

<table>
<tr><td></td><td>економічних показників</td><td colspan="2">автоматизації налаштування сценаріїв, запуску атак та обробки результатів.
2) Підвищення об’єктивності та точності оцінки стійкості порівняно з ручним аналізом логів та трафіку.
3) Відсутність витрат на ліцензійне програмне забезпечення (використання Open Source-інструментів).</td></tr>
<tr><td rowspan="11">7.</td><td rowspan="11">Стадії і етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td>02.09.2024 - 02.10.2024</td><td>Затвердження теми роботи та керівника. Попереднє формування вимог до тестового IoT-полігону та переліку базових мережевих атак.</td></tr>
<tr><td>03.09.2024 - 24.10.2024</td><td>Аналіз і пошук літератури щодо безпеки IoT, типових мережевих атак та методів оцінки стійкості; огляд стандартів та рекомендацій з кібербезпеки IoT.</td></tr>
<tr><td>25.10.2024 - 09.11.2024</td><td>Розробка концепції архітектури arduino-базованого тестового полігону (топологія мережі, ролі вузлів, вибір протоколів та інструментів моніторингу).</td></tr>
<tr><td>10.11.2024 - 24.11.2024</td><td>Реалізація базового програмного забезпечення для IoT-вузлів (скетчі Arduino/ESP), налаштування брокера MQTT та служб мережевої інфраструктури.</td></tr>
<tr><td>25.11.2024 - 08.12.2024</td><td>Розробка та апробація сценаріїв базових мережевих атак (MITM, TCP/UDP-Flood, сканування портів) з використанням спеціалізованих інструментів.</td></tr>
<tr><td>09.12.2024 - 29.01.2025</td><td>Створення модулів автоматизованого збору, агрегації та попередньої обробки експериментальних даних (PCAP, логи брокера й вузлів).</td></tr>
<tr><td>30.01.2025 - 28.02.2025</td><td>Розробка модулів оцінки стійкості: розрахунок метрик якості виявлення атак, побудова графічних звітів.</td></tr>
<tr><td>01.03.2025 - 01.04.2025</td><td>Реалізація та тестування механізмів підвищення стійкості (HMAC-підписTLS-тунелювання, фільтрація й rate-limiting) у межах тестового полігону.</td></tr>
<tr><td>01.05.2025 - 30.08.2025</td><td>Проведення серії експериментів, порівняння конфігурацій «без захисту» та «із захистом», аналіз результатів, підготовка чорнових таблиць і графіків.</td></tr>
<tr><td>01.09.2025 - 30.10.2025</td><td>Оформлення звіту про науково-дослідну практику. Написання статті за матеріалами кваліфікаційної роботи.</td></tr>
</table>

		10.10.2025 - 30.10.2025	Підготовка і оформлення основних розділів пояснювальної записки (вступ, аналітичний огляд, опис архітектури та експериментальної методики, аналіз результатів).
		10.10.2025 - 30.11.2025	Оформлення пояснювальної записки кваліфікаційної роботи відповідно до вимог до звітів про НДР. Оформлення списку використаних джерел.
		01.11.2025 - 30.11.2025	Підготовка і оформлення звітних матеріалів та додатків кваліфікаційної роботи (лістинги коду, акти впровадження, інструкції).
		24.11.2025 - 30.11.2025	Оформлення звіту про переддипломну практику.
		24.11.2025 - 30.11.2025	Представлення кваліфікаційної роботи керівнику та рецензенту.
8.	Порядок контролю і приймання програмного продукту	1) Перевірку ходу розробки виконувати згідно з календарним планом. 2) Захист розробленої системи провести на засіданні Екзаменаційної комісії. 3) Пояснювальну записку подати на паперових носіях та в електронному вигляді згідно з вимогами ВНЗ.	

Виконавець:

студент групи КУ-61

Шульга Р.В.

Замовник:

к. т. н., доцент

Хруслов М. М.

Додаток В

Програма і методика випробувань

програмного виробу

«Програмно-апаратний комплекс для оцінки та підвищення кіберстійкості Arduino-базованих IoT-вузлів»

1. **Об'єкт випробувань**
    1. Назва програмного виробу: «Програмно-апаратний комплекс для оцінки та підвищення кіберстійкості Arduino-базованих IoT-вузлів».
    2. Галузь застосування: Кібербезпека Інтернету речей (IoT), автоматизовані системи управління технологічними процесами (АСУ ТП), навчальний процес.
    3. Відомості запозичуються з відповідних розділів Технічного завдання
2. **Мета випробувань.** Перевірка відповідності функціональності програмно-апаратної реалізації комплексу вимогам, заявленим в Технічному завданні, а саме: здатності системи виявляти мережеві атаки, забезпечувати цілісність даних та підтримувати працездатність в умовах перевантаження (DoS).
3. **Загальні положення**
    1. **Підстави для проведення випробувань:** Підставою для проведення випробувань є наказ про призначення атестаційної комісії.
    2. **Місце і тривалість випробувань:** Приймальні випробування проводяться на базі лабораторії кафедри КСР в період роботи атестаційної комісії.
    3. **Обсяг випробувань:** Приймальні випробування програмного виробу проводяться в обсязі, відповідному цій програмі і методиці випробувань.
    4. **Організації, які беруть участь у випробуваннях:** Приймальні випробування проводяться атестаційною комісією за участю

Замовника (керівника), Виконавця (студента) та інших осіб, присутніх на засіданні.

4. **Вимоги до програми або програмного виробу**. Комплекс повинен задовольняти наступним вимогам:
    1. Забезпечувати збір телеметрії з датчика DHT11 та її передачу через Wi-Fi (ESP8266) за протоколом MQTT.
    2. Реалізовувати автентифікацію кожного повідомлення за алгоритмом HMAC-SHA256.
    3. Забезпечувати захист від DoS-атак на рівні прошивки (Token Bucket), підтримуючи PDR > 85% при навантаженні 500 пакетів/с.
    4. Автоматично відхиляти пакети з невірним цифровим підписом або застарілим лічильником (Nonce).
    5. Виявляти спроби ARP-спуфінгу та аномальної активності за допомогою IDS Suricata.
    6. Коректно працювати в середовищі Linux (Server) та на мікроконтролері ATmega328P.
    7. Візуалізувати метрики роботи в реальному часі (InfluxDB/Grafana або консоль).
5. **Вимоги до програмної документації**. Програмною документацією до виробу вважати:
    1. Технічне завдання на розробку (Додаток Б до пояснювальної записки);
    2. Дану Програму і методику випробувань (Додаток В до пояснювальної записки);
    3. Інструкцію з налаштування та експлуатації комплексу (Розділ 3 пояснювальної записки);
    4. Лістинги вихідного коду (Додаток А).
6. **Засоби і порядок випробувань**

6.1. **Засоби випробувань.** Для проведення випробувань необхідний експериментальний стенд у складі:

- IoT-вузол: Arduino Uno + ESP8266 + DHT11;
- Сервер: ПК або Raspberry Pi з ОС Linux (Ubuntu/Debian), встановленими Mosquitto, Suricata, Python 3.9+;
- Атакуюча станція: Ноутбук з ОС Kali Linux та інструментами hping3, Bettercap, Wireshark;
- Wi-Fi маршрутизатор.

6.2. Порядок проведення випробувань: Випробування проводяться в два етапи: ознайомчий (перевірка документації та комплектації); випробування функціональності (виконання тест-кейсів).

Перелік перевірок (функціональні тести):

Тест 1. Перевірка штатного режиму роботи (Baseline)

1. Запустити брокер Mosquitto та скрипт верифікації на сервері.
2. Увімкнути живлення IoT-вузла.
3. Перевірити лог сервера: очікується успішне підключення пристрою та надходження валідних даних температури/вологості кожну секунду.
4. Результат: Дані надходять, підпис (HMAC) перевіряється успішно (OK).

Тест 2. Перевірка захисту від підміни даних (MITM / Tampering)

1. На станції Kali Linux запустити Bettercap, активувати ARP-спуфінг.
2. Перехопити MQTT-пакет, змінити значення температури у payload і відправити брокеру.
3. Перевірити реакцію сервера.
4. Результат: Сервер відхиляє пакет з повідомленням "Signature Mismatch" або "Invalid HMAC". Дані в БД не записуються.

Тест 3. Перевірка стійкості до DoS-атаки (Rate Limiting)

1. На станції Kali Linux запустити генератор трафіку hping3 (режим UDP Flood, 500 pkt/s) на IP-адресу Arduino.
2. Спостерігати за світлодіодом індикації на платі (має сигналізувати про відкидання пакетів) та логом сервера.

3. Результат: Пристрій не зависає, продовжує відправляти телеметрію (можливі збільшені затримки), коефіцієнт PDR залишається в межах норми (>85%).

Тест 4. Перевірка роботи IDS Suricata

1. Виконати сканування портів або ARP-спуфінг в мережі.
2. Перевірити файл журналу /var/log/suricata/fast.log.
3. Результат: У журналі з'являються записи про виявлені загрози (наприклад, "ARP Spoofing Detected" або "Potential MQTT DoS").

Висновки: Тест 1 успішно пройшов випробування (система функціонує). Тест 2 успішно пройшов випробування (цілісність забезпечено). Тест 3 успішно пройшов випробування (відмови в обслуговуванні не сталося). Тест 4 успішно пройшов випробування (атаки виявлено). Комплекс визнано таким, що пройшов випробування.

Виконавець: студент групи КУ-51, Шульга Р.В.