Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

Розробка апаратного месенджера для захисту ведення конфіденційних переговорів на базі мікроконтролерної платформи

Виконав: студент 4 курсу, гр. 1КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Кравчук В.О.
(прізвище та ініціали)

Керівник: к.ф.-м.н., доц., доцент каф. МБІС
Шиян А.А.
(прізвище та ініціали)
«____» _____ 2025 р.

Рецензент: к.ф.-м.н., доц., доцент каф. ОТ
Крупельницький Л. В.
(прізвище та ініціали)
«____» _____ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“_____” _____2025р.

Вінниця ВНТУ – 2025

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

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

д.т.н., проф. Яремчук Ю.Є.
«20» березня 2025 р.

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Кравчуку Владиславу Олександровичу
(прізвище, ім`я, по-батькові)

1. Тема роботи: Розробка апаратного месенджера для захисту ведення конфіденційних переговорів на базі мікроконтролерної платформи
Керівник роботи к.ф.-м.н., доц. каф. МБІС Шиян А.А.
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "20" березня 2025 року №97
2. Термін подання студентом роботи"2" червня 2025 року
3. Вихідні дані до роботи:Метою роботи є здійснення розробки апаратного месенджера для захищеного ведення конфіденційних переговорів на базі мікроконтролерної платформи з використанням технологій локального бездротового зв'язку, базових криптографічних алгоритмів та мобільного інтерфейсу для зручної взаємодії з користувачем.
4. Зміст текстової частини:У першому розділі розглядаються теоретичні основи побудови захищених систем зв'язку, проаналізовано актуальні загрози інформаційній безпеці в цифрових каналах та сучасні програмні й апаратні засоби захищеної комунікації. В другому розділі виконується проектування архітектури захищеного месенджера на базі мікроконтролера ESP32. У третьому розділі описано практичну реалізацію всіх компонентів системи: налаштування SoftAP та WebSocket-сервера на ESP32,

реалізація алгоритмів шифрування та OTP-автентифікації, взаємод
мобільними застосунками. Проведено демонстраційне тестування систем
імітацією реальних сценаріїв: підключення, оновлення ключів, пере
повідомлень, обробка помилок. Зафіксовано результати тестування, провед
аналіз продуктивності та сформульовано висновки щодо ефективності
надійності запропонованого рішення.

5. Перелік ілюстративного матеріалу:

У першому розділі бакалаврської кваліфікаційної роботи наявно 2 рису
5 таблиці; у другому розділі – 6 рисунків, 3 таблиці; у третьому розділі -
рисунків, 1 таблиця.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
I	Шиян А. А., к.ф.-м.н., доц. каф. МБІС		
II	Шиян А. А., к.ф.-м.н., доц. каф. МБІС		
III	Шиян А. А., к.ф.-м.н., доц. каф. МБІС		

7.Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Приміт
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025	Викон
2	Аналіз предметної області обраної теми	24.03.2025	13.04.2025	Викон
3	Розробка, реалізація	14.04.2025	27.04.2025	Викон
4	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025	Викон
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025	Викон
6	Виправлення, уточнення, корегування бакалаврської кваліфікаційної роботи	28.05.2025	08.06.2025	Викон
7	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025	Викон

Студент _____ (підпис) Ткачук В.О. (прізвище та ініціали)

Керівник роботи _____ (підпис) Шиян А.А. (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 100 сторінок формату А4, на яких наведено 22 рисунків, 6 таблиць, список використаних джерел містить 62 найменувань.

Метою роботи є створення автономної системи обміну повідомленнями на основі мікроконтролера ESP32 із використанням сучасних криптографічних методів (AES-128, TOTP), що дозволяє забезпечити конфіденційність, цілісність та автентичність переданої інформації. У роботі проаналізовано наявні програмні й апаратні рішення у сфері захищеної комунікації, обґрунтовано вибір архітектури системи, апаратної платформи, протоколів обміну даними та методів шифрування. Описано розробку програмного забезпечення для ESP32, мобільних додатків для клієнтів і адміністратора, а також реалізовано механізми автентифікації користувачів, обміну конфігураціями та віддаленого керування. Проведено тестування функціональності системи, визначено її обмеження та перспективи подальшого розвитку. Результати розробки можуть бути використані у сфері захисту інформації, зокрема в урядових, оборонних, корпоративних та експедиційних умовах, де потрібна автономна, ізольована та надійна комунікація.

Ключові слова: захищена комунікація, ESP32, AES, TOTP, шифрування, мобільний застосунок, WebSocket, SoftAP, автентифікація, інформаційна безпека.

ANNOTATION

The bachelor's qualification work consists of 100 pages of A4 format, which contains 22 figures, 6 tables, the list of references contains 62 items.

The goal of the project is to create an autonomous message exchange system based on the ESP32 microcontroller, using modern cryptographic methods (AES-128, TOTP) to ensure confidentiality, integrity, and authenticity of transmitted data. The work analyzes existing software and hardware solutions in the field of secure communication, justifies the choice of system architecture, hardware platform, data exchange protocols, and encryption algorithms. It describes the development of software for the ESP32, mobile applications for clients and administrators, and the implementation of authentication mechanisms, configuration exchange, and remote control. System functionality was tested, and its limitations and potential for further development were identified. The developed solution can be applied in the field of information security, particularly in governmental, defense, corporate, and fieldwork scenarios where autonomous, isolated, and reliable communication is required.

Keywords: secure communication, ESP32, AES, TOTP, encryption, mobile application, WebSocket, SoftAP, authentication, information security.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасних умовах зростання кіберзагроз і активного використання цифрових засобів комунікації, питання забезпечення конфіденційності інформаційного обміну набуває особливої актуальності. Багато сфер діяльності, зокрема політична, бізнесова, військова, дипломатична, потребують гарантованого захисту перемовин від зовнішнього втручання або прослуховування. Існуючі програмні засоби обміну повідомленнями переважно базуються на централізованій інфраструктурі, що створює потенційні точки вразливості як у мережевому трафіку, так і в самій платформі.

Актуальність даної теми обумовлена потребою у створенні апаратного рішення для конфіденційного спілкування, яке не залежить від сторонніх серверів чи Інтернет-з'єднання, функціонує у межах локальної мережі та забезпечує базовий рівень шифрування повідомлень. Саме апаратна реалізація на основі мікроконтролерної платформи дозволяє зменшити кількість потенційних векторів атак, обмежити коло учасників комунікації та підвищити загальну безпеку системи.

Мета і задачі дослідження.

Метою даної роботи є розробка апаратного месенджера для захисту ведення конфіденційних переговорів на базі мікроконтролерної платформи.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– проаналізувати сучасні підходи до побудови автономних захищених систем зв'язку та обґрунтувати вибір апаратної платформи і криптографічних засобів;

– спроектувати архітектуру системи з підтримкою базових механізмів шифрування та забезпечити ідентифікацію учасників комунікації;

− реалізувати мобільний застосунок і провести тестування працездатності та захищеності системи.

Об’єктом дослідження. Є процес безпечної передачі текстових повідомлень у межах замкненого бездротового середовища.

Предметом дослідження. Виступають методи побудови апаратного месенджера з підтримкою базових механізмів шифрування, автентифікації та обміну даними.

Новизна роботи. Новизна роботи полягає у поєднанні автономної мікроконтролерної платформи, локального бездротового обміну та мобільного інтерфейсу для створення захищеного месенджера, що не залежить від глобальної мережі Інтернет.

Практична цінність розробки. Практична цінність розробки полягає у можливості її використання як засобу для проведення конфіденційних переговорів у закритих середовищах – наприклад, у військових, корпоративних чи екстрених ситуаціях, де немає довіри до сторонніх цифрових сервісів або доступу до мережі. Такий підхід дозволяє забезпечити базовий рівень інформаційної безпеки при мінімальних апаратних затратах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНИХ РІШЕНЬ У СФЕРІ ЗАХИЩЕНОГО ОБМІНУ ПОВІДОМЛЕННЯМИ

У першому розділі буде розглянуто теоретичні основи побудови захищених систем зв’язку, проаналізовано актуальні загрози інформаційній безпеці в цифрових каналах та сучасні програмні й апаратні засоби захищеної комунікації. Окрема увага приділятиметься аналізу криптографічних алгоритмів та їх придатності для використання у вбудованих системах. На основі аналізу сформульовано завдання на розробку.

1.1 Особливості побудови захищених систем комунікації

У процесі проектування будь-якої системи зв’язку, особливо тієї, що призначена для передавання конфіденційної інформації, першочерговим завданням є дотримання фундаментальних принципів інформаційної безпеки. У фаховій літературі вони часто згруповані в абревіатуру CIA (Confidentiality, Integrity, Availability), або ж КЦД [1]:

– конфіденційність – забезпечення недоступності інформації для осіб, які не мають відповідного доступу. У комунікаційних системах це досягається шляхом використання шифрування даних, а також автентифікації користувачів;

– цілісність – гарантування того, що передана або збережена інформація не була змінена чи зіпсована неналежним чином. Реалізується через контрольні суми, хеш-функції, цифрові підписи;

– доступність – можливість своєчасного і безперешкодного доступу до інформації уповноваженими особами. Це включає як технічну справність систем, так і протидію атакам типу DoS (відмова в обслуговуванні);

Іноді до основних додають ще два принципи:

– автентичність – підтвердження достовірності джерела інформації (хто саме надіслав дані).

– незаперечність – забезпечення неможливості відмови від авторства повідомлення з боку відправника (важливо для юридичних транзакцій) [2].

Ці принципи закладено у стандартах безпеки, зокрема у ISO/IEC 27001, який є базовим міжнародним стандартом з управління інформаційною безпекою.

У захищених месенджерах ці принципи реалізуються через комплекс заходів: наскрізне шифрування (end-to-end encryption) гарантує конфіденційність, цифрові підписи – цілісність і автентичність, а резервне дублювання каналів – доступність. Значення кожного принципу та приклад реалізації продемонстровано на (табл.1.1) [3].

Таблиця1.1 – Схема фундаментальних принципів інформаційної безпеки.

Основні принципи інформаційної безпеки		
Принципи	Значення	Приклад реалізації
Конфіденційність	Шифрування повідомлень	AES-128, RSA
Цілісність	Гарантія незмінності	HMAC, SHA-256
Доступність	Безперебійна робота	Захист від DoS
Автентичніссть	Підтвердження джерела	Цифровий підпис
Незаперечність	Неможливість заперечити авторство	PKI + підпис

Як видно з (табл. 1.1), кожен фундаментальний принцип безпеки реалізується за допомогою відповідних технологічних рішень. Так, конфіденційність забезпечується шифруванням, цілісність – механізмами хешування та цифрових підписів, а доступність – резервуванням систем. Важливо розуміти, що ці принципи є взаємопов'язаними, і побудова по-справжньому захищеної системи вимагає комплексного підходу, що враховує всі три аспекти. Саме на порушення цих базових принципів і спрямовані загрози інформаційній безпеці. Залежно від способу впливу загрози поділяються на пасивні та активні [4]:

– пасивні загрози полягають у спостереженні за передачею без активного втручання (наприклад, перехоплення трафіку або зняття електромагнітних випромінювань). Основна мета – здобуття конфіденційної інформації;

– активні загрози передбачають зміну інформації або структури мережі: це можуть бути атаки типу "людина посередині" (MITM), підміна повідомлень, нав’язування шкідливого програмного коду.

З точки зору походження, загрози бувають [5]:

– зовнішні – ініційовані ззовні (хакерські атаки, сканування портів, тощо);

– внутрішні – походять від користувачів, які мають доступ до системи, але використовують його зловмисно або необережно.

Найбільш поширені загрози в системах комунікації [6]:

– перехоплення повідомлень – можлива через незашифровані канали;

– атака повторного відтворення (Replay attack) – зловмисник записує та пізніше відтворює повідомлення, видаючи його за нове;

– атака MITM (Man-in-the-middle) – зловмисник перехоплює та змінює повідомлення між двома сторонами;

– фішинг та соціальна інженерія – психологічний вплив на користувача для отримання паролів або доступу;

– атаки типу DoS/DDoS – спрямовані на виведення з ладу системи обміну повідомленнями.

Захист від таких загроз потребує комплексного підходу: використання шифрування, автентифікації, перевірки цілісності та моніторингу аномалій у поведінці системи [7].

Шифрування забезпечує конфіденційність переданої інформації, перетворюючи її в нечитабельний формат для неавторизованих осіб. Існують два основні типи шифрування [8]:

– симетричне шифрування. Використовує один ключ для шифрування та дешифрування. Прикладом є алгоритм AES (Advanced Encryption Standard), який широко застосовується в сучасних системах захисту інформації [9].

– асиметричне шифрування. Використовує пару ключів – публічний та приватний. Алгоритми RSA та ECC (Elliptic Curve Cryptography) є прикладами асиметричного шифрування, що забезпечує високий рівень безпеки при передачі даних [10].

Аутентифікація підтверджує особу користувача, а авторизація визначає його права доступу. Методи аутентифікації включають [11]:

– однофакторна аутентифікація - використання одного фактора, наприклад, пароля;

– двофакторна аутентифікація (2FA) - поєднання двох факторів, наприклад, пароля та одноразового коду, надісланого на мобільний пристрій;

– багатофакторна аутентифікація (MFA) - використання трьох або більше факторів, включаючи біометричні дані.

Забезпечення цілісності даних передбачає виявлення будь-яких змін у переданій інформації. Основні методи [12]:

– хеш-функції. Обчислення контрольної суми повідомлення для виявлення змін;

– цифрові підписи. Використання криптографічних методів для підтвердження автентичності та цілісності повідомлення.

Для запобігання атакам MITM використовуються :

– протоколи захищеного з’єднання, наприклад, TLS (Transport Layer Security), який забезпечує шифрування та аутентифікацію сторін [13].

– сертифікати цифрового підпису. Підтверджують автентичність серверів та клієнтів;

VPN створює захищене з’єднання через публічні мережі, забезпечуючи конфіденційність та цілісність переданих даних [14].

Незважаючи на широке застосування програмних рішень для захисту комунікацій, важливим залишається питання довіри до інфраструктури, яка забезпечує ці рішення. Зокрема, централізовані сервери обміну повідомленнями, хмарні сховища ключів або маршрутизатори можуть бути скомпрометованими як через зовнішні атаки, так і через внутрішні ризики – помилки адміністрування, цілеспрямовані дії інсайдерів або урядовий контроль.

В умовах обмеженої довіри до інтернет-інфраструктури зростає актуальність автономних засобів зв’язку, які не покладаються на глобальні сервіси. Такий підхід дозволяє забезпечити ізольовану безпечну комунікацію, наприклад, у польових умовах, під час критичних переговорів або в зонах обмеженого доступу до мережі.

Більшість популярних програмних рішень функціонують у рамках універсальних операційних систем (наприклад, Android або iOS), які самі по собі можуть містити вразливості. Крім того, ці рішення залежать від своєчасного оновлення, наявності доступу до інтернету, а також програмного коду закритого типу, який унеможливлює незалежну верифікацію безпеки [15].

У випадках, коли необхідно досягти максимального контролю над каналом зв'язку, виникає потреба у створенні власного ізольованого рішення, що функціонує поза межами звичних програмних платформ. Однією з відповідей на виклики сучасної цифрової безпеки є застосування апаратних рішень, побудованих на основі вбудованих систем (embedded systems). Вони дозволяють реалізувати критично важливу функціональність (шифрування, автентифікацію, обмін повідомленнями) в ізольованому середовищі, контрольованому розробником, без необхідності залучення сторонніх служб.

Таким чином, апаратний підхід до захищеної комунікації не лише розширює можливості у забезпеченні конфіденційності, а й підвищує довіру до самої системи за рахунок прозорості, автономності та незалежності від сторонніх інфраструктур.

1.2. Огляд програмних засобів захищеного обміну повідомленнями

У сучасному цифровому просторі велика увага приділяється питанням забезпечення конфіденційності спілкування користувачів. Існує низка програмних рішень, що забезпечують обмін повідомленнями з використанням криптографічних методів захисту. До найвідоміших із них належать Signal, Telegram, Threema, Session, Element та інші [16].

Месенджер Signal вважається еталонним прикладом застосування наскрізного шифрування (end-to-end encryption, E2EE). Його протокол Signal Protocol використовує комбінацію X3DH (Extended Triple Diffie-Hellman) для встановлення сеансу та Double Ratchet для безперервного оновлення ключів. Ці особливості забезпечують приватність пересилки повідомлень, навіть якщо ключі скомпрометовано після завершення сеансу – концепція forward secrecy.

Signal має відкритий вихідний код і використовується як основа шифрування у WhatsApp і Facebook Messenger [17-18].

Threema – платний месенджер, який не вимагає номера телефону чи email для реєстрації. Всі повідомлення захищені E2EE, а зберігання метаданих мінімізовано. Протокол Threema базується на відкритих криптографічних бібліотеках, зокрема NaCl [19].

Session – месенджер, орієнтований на анонімність і децентралізацію. Його архітектура базується на мережі Oxen (форк Monero), а обмін повідомленнями здійснюється без потреби в телефонному номері. Повідомлення маршрутизуються через кілька вузлів, аналогічно до принципів Tor, а ключі зберігаються тільки локально [20].

Element – клієнт для децентралізованого протоколу Matrix, який дозволяє реалізовувати шифровані групові чати та інтеграцію з іншими системами. Використовується Olm/Megolm протокол, що підтримує наскрізне шифрування для приватних і групових повідомлень [21].

Для наочного порівняння ключових характеристик розглянутих програмних засобів наведемо узагальнюючу (табл. 1.2) [22].

Таблиця 1.2 – Порівняння месенджерів та їх підходів

Месенджер	E2EE за замовчуванням	Анонімність	Відкритий код	Децентралізація	Потреба в номері телефону
Signal	Так	Ні	Так	Ні	Так
Telegram	Ні (тільки секретні)	Частково	Частково	Ні	Так
Threema	Так	Так	Частково	Ні	Ні
Session	Так	Так	Так	Так	Ні
Element	Так	Частково	Так	Так	Ні

Як, видно, з (табл. 1.2), незважаючи на високий рівень захисту, більшість популярних програмних месенджерів мають суттєві архітектурні відмінності. Такі рішення як Signal і Telegram, хоч і є популярними, залишаються централізованими та вимагають прив'язки до номера телефону, що зменшує анонімність користувача.

Месенджери Session та Element пропонують децентралізовану архітектуру, що підвищує стійкість до блокувань, проте вони все ще залежать від доступності глобальної мережі Інтернет.

Таким чином, попри наявність потужних програмних засобів, вони мають спільну вразливість — залежність від централізованої або глобальної інфраструктури та інтернет-з'єднання. Це робить їх непридатними для використання в умовах повної інформаційної ізоляції або за відсутності довіри до зовнішніх мереж, що й обґрунтовує актуальність розробки автономних апаратних рішень.

Попри високий рівень захисту, більшість програмних месенджерів залишаються вразливими до:

– залежності від інтернет-з'єднання та центральних серверів, які можуть бути відключені або атаковані (Signal, Telegram);

– можливості знеособлення користувача, якщо використовуються номери телефонів (Signal, Telegram);

– збереження частини метаданих на стороні сервера (навіть при використанні E2EE).

Таким чином, хоча програмні засоби мають високий рівень захисту, вони часто залежать від централізованої інфраструктури та інтернет-з'єднання. Це робить їх вразливими до атак, цензури або технічних збоїв. У цьому контексті апаратні рішення можуть забезпечити альтернативу, що працює в умовах обмеженого або локального середовища, без необхідності глобального доступу.

1.3 Апаратні рішення в сфері захищеної комунікації

Зважаючи на зростаючі загрози у сфері цифрової безпеки, останніми роками спостерігається підвищений інтерес до апаратних рішень, які забезпечують захищений обмін повідомленнями незалежно від традиційних мережевих інфраструктур. Розглянемо приклади реалізації таких рішень – як на базі мікроконтролерних платформ, так і комерційні рішення спеціального призначення.

Для створення низьковартісних та автономних систем комунікації використовуються здебільшого ESP32, Arduino, Raspberry Pi.

ESP32 – потужний мікроконтролер з вбудованим Wi-Fi та Bluetooth. Завдяки наявності достатньої кількості обчислювальних ресурсів, він здатен виконувати криптографічні операції (AES, RSA, SHA) у реальному часі, що робить його ідеальним для розробки захищених чатів, IoT-рішень, автономних систем обміну повідомленнями тощо [23].

Arduino – менш продуктивна платформа, але широко використовується в освітніх проєктах. Для обміну повідомленнями зазвичай потребує додаткових модулів зв'язку (Wi-Fi, GSM) і зовнішньої криптографічної підтримки.

Raspberry Pi – мінікомп'ютер, що дозволяє запускати повноцінні ОС (наприклад, Linux) і програмне забезпечення для шифрування (GnuPG, Matrix-сервери тощо). Часто використовується в поєднанні з сенсорними екранами для створення користувацьких інтерфейсів. Зовнішній вигляд та ключові відмінності цих популярних платформ для прототипування зображено на (рис. 1.1). [24].

Рисунок 1.1 – Зовнішній вигляд сучасних мікроконтролерних платформ

Як видно з (рис. 1.1), платформи суттєво відрізняються за форм-фактором та компонуванням. Raspberry Pi є повноцінним одноплатним комп'ютером з роз'ємами HDMI, USB, Ethernet, що робить його надлишковим для простих завдань, але зручним для складних систем. Натомість ESP32 вирізняється високим ступенем інтеграції (процесор, пам'ять, Wi-Fi/Bluetooth модуль на одній платі), що ідеально підходить для створення компактних автономних пристроїв, що і є завданням даної роботи. Платформа Arduino Uno, в свою чергу, є найпростішою і вимагає додаткових модулів (шилдів) для мережевої взаємодії.

У відкритому доступі існує низка проєктів, які демонструють концепції захищеної комунікації. Ці розробки слугують прикладами реалізації безпечного обміну повідомленнями у середовищах з обмеженим або відсутнім доступом до інтернету, а також підкреслюють важливість використання сучасних методів шифрування у побудові захищених комунікаційних каналів.

Одним із таких проєктів є Meshtastic – децентралізована мережа, що використовує мікроконтролери ESP32 у поєднанні з радіомодулями LoRa для створення автономних каналів обміну повідомленнями без необхідності підключення до мережі Інтернет. Проєкт дозволяє формувати mesh-мережу, у якій кожен пристрій виступає вузлом ретрансляції, що підвищує дальність і надійність зв’язку. Його відкритий вихідний код дає змогу адаптувати систему під конкретні сценарії використання [25].

PineDio Stack представляє собою ще одну open-source розробку, яка надає інструменти для створення зашифрованої mesh-мережі на базі Pine64-пристроїв. Платформа орієнтована на забезпечення конфіденційності переданої інформації та може бути використана в критичних ситуаціях, наприклад, під час природних катастроф або в умовах інформаційної ізоляції [26].

Off-The-Record ESP32 Messenger – це експериментальний прототип месенджера, створений на базі мікроконтролера ESP32. Він реалізує наскрізне шифрування (E2EE) із використанням алгоритму AES і WebSocket-з’єднання для передавання повідомлень. Основна мета цього проєкту – демонстрація можливості

створення захищених комунікаційних каналів навіть на базі недорогих апаратних платформ [27].

Окрім експериментальних та хобі-проєктів, існує також ряд спеціалізованих пристроїв, які були розроблені для задоволення потреб державних структур, військових та корпоративного сектору в захищеній комунікації. Такі рішення поєднують у собі як апаратні, так і програмні засоби захисту, що забезпечують високий рівень конфіденційності. Кілька прикладів таких пристроїв наведено на (рис. 1.2) [28-29].

Рисунок 1.2 – Спеціалізовані комерційні пристрої захищеної комунікації

Як можна бачити на (див. рис. 1.2), ці пристрої часто мають вигляд звичайних смартфонів, однак їх ключова відмінність полягає у модифікованій операційній системі та вбудованих апаратних модулях безпеки.

Одним із найбільш відомих комерційних прикладів є Blackphone від компанії Silent Circle. Це смартфон, що працює під керуванням спеціального Android-

дистрибутиву Silent OS. Він підтримує шифрування голосових дзвінків, текстових повідомлень і даних, а також надає користувачеві гнучкі інструменти контролю над приватністю [30].

PGP-телефони являють собою клас спеціалізованих пристроїв, які є не стільки унікальною апаратною розробкою, скільки комерційними смартфонами (історично — Blackberry, сьогодні — кастомізовані версії Android) з глибоко інтегрованим програмним забезпеченням для шифрування даних за протоколом Pretty Good Privacy (PGP). Основне призначення таких телефонів — забезпечення наскрізного шифрування електронної пошти, що дозволяє безпечно передавати конфіденційні документи та вести листування, використовуючи асиметричну криптографію. Часто такі пристрої піддаються додатковому «загартуванню»: з них видаляються сервіси Google, обмежується встановлення сторонніх додатків, а іноді навіть фізично видаляються камери та мікрофони. Через таку вузьку спеціалізацію та орієнтацію на максимальну безпеку, вони знайшли свою нішу в урядових установах, корпоративному секторі та серед журналістів, які працюють з чутливою інформацією [31].

Sectéra Edge, розроблений компанією General Dynamics, є сертифікованим урядом США пристроєм для передачі конфіденційної інформації. Пристрій підтримує шифрування голосового зв’язку, передавання даних та обміну текстовими повідомленнями. Завдяки відповідності державним стандартам безпеки, він активно використовується у сфері національної безпеки [32].

Bittium Tough Mobile – це приклад високонадійного смартфона, розробленого для військово-промислового застосування. Він має багаторівневу систему захисту, яка охоплює як апаратні, так і програмні компоненти. Серед ключових особливостей – контроль за цілісністю середовища, можливість шифрування з використанням сертифікованих алгоритмів та наявність спеціальних режимів роботи для запобігання витоку даних [33].

Для узагальнення та наочного порівняння розглянутих комерційних рішень, їх ключові характеристики зведено у (табл. 1.3) [32].

Таблиця 1.3 – Порівняльна таблиця комерційних апаратних рішень

Пристрій	Шифрування	ОС / Платформа	Призначення	Особливості
Blackphone 2	End-to-End (Silent Phone)	Silent OS (Android)	Бізнес/ приватність	Фокус на приватність, відкритий код
Bittium Tough Mobile 2	AES, VPN, TPM	Android Secure Platform	Військове, урядове використання	Сертифікація, захист на рівні апаратури
Sectéra Edge	AES, Suite B	Windows CE	Уряд США, спецслужби	Підтримка Top Secret даних
PGP-телефони	PGP, OTR	Змінювана	Бізнес, спецслужби	Часто кастомізовані, продаються через дистриб’юторів

З аналізу (табл. 1.3) випливає, що спеціалізовані апаратні засоби забезпечують високий рівень захисту завдяки використанню сертифікованих алгоритмів та апаратних модулів безпеки (TPM). Водночас вони орієнтовані на вузькоспеціалізовані ринки (урядовий, військовий), що зумовлює їх високу вартість та обмежену доступність для широкого кола користувачів.

Тепер, коли ми розглянули конкретні приклади, доцільно провести узагальнене порівняння переваг та недоліків програмного та апаратного підходів в цілому. Результати такого порівняння наведено у (табл. 1.4) [34].

Таблиця 1.4 – Порівняльна характеристика підходів захищеної комунікації

Параметр	Програмні рішення	Апаратні рішення
Залежність від Інтернету	Так (більшість)	Ні (у випадку автономних мереж)
Вартість	Низька/безкоштовна	Відносно висока
Гнучкість	Висока	Залежить від реалізації
Безпека ключів	Можливий витік	Можна ізолювати фізично
Застосування в умовах НС	Обмежене	Можливе (локальні радіомережі)

Як демонструє (таб. 1.4), програмні рішення є значно дешевшими та гнучкішими, однак їхня залежність від Інтернету та ризики, пов'язані з можливим витоком ключів на універсальних пристроях, є суттєвими недоліками. Натомість апаратні рішення, попри вищу вартість, демонструють фундаментальну перевагу в автономності та можливості фізичної ізоляції криптографічних ключів. Саме ця властивість робить їх незамінними та більш надійними для сценаріїв, що вимагають максимальної конфіденційності та стійкості до зовнішніх впливів.

Таким чином, апаратні рішення демонструють вищу автономність, контрольованість і стійкість до зовнішніх впливів, що робить їх актуальними в умовах обмеженого або відсутнього доступу до традиційних мереж.

З аналізу засобів, проведеного вище, можемо бачити, що комерційні апаратні засоби захищеної комунікації часто мають високу вартість порівняно з програмними або відкритими альтернативами. Це пояснюється такими факторами:

– інтеграція апаратного шифрування, модулів TPM, механізмів захисту ключів;
– сертифікація згідно з державними або міжнародними стандартами безпеки (наприклад, FIPS, NATO, Common Criteria);
– закриті розробки та обмежене виробництво, спрямоване на специфічну цільову аудиторію – державні органи, військові структури, великі корпорації;
– вартість супроводу: багато пристроїв продаються разом із сервісними пакетами (підтримка, оновлення, ліцензії на ПЗ).

Через це масове впровадження таких засобів у повсякденне користування залишається обмеженим. Саме тому виникає потреба в розробці власних, доступних рішень на базі мікроконтролерів, що і є метою даного проекту.

1.4. Аналіз криптографічних методів для вбудованих систем

Однією з ключових складових побудови захищеної системи комунікації є вибір належного криптографічного алгоритму, здатного забезпечити конфіденційність, цілісність і автентичність переданих даних. Особливе значення

це питання набуває при розробці на базі мікроконтролерних платформ, де ресурси системи значно обмежені – як обчислювальні, так і енергетичні.

Криптографічні алгоритми умовно поділяються на [35-36]:

– симетричні алгоритми шифрування, де для шифрування та дешифрування використовується один і той самий ключ. Типові представники – AES, ChaCha20, DES, RC4. Їх основними перевагами є висока швидкодія та відносно низьке споживання ресурсів;

– асиметричні алгоритми шифрування, які оперують парою відкритого і закритого ключів. Вони дозволяють реалізовувати цифрові підписи та безпечну передачу ключів. Найбільш відомі приклади – RSA, ElGamal, ECC (Elliptic Curve Cryptography). Головний недолік – високе навантаження на ресурси, що не завжди прийнятне для вбудованих пристроїв.

Для реалізації криптографічних алгоритмів у мікроконтролерних системах слід враховувати наступні обмеження:

– обмежений обсяг оперативної пам’яті (наприклад, ESP32 має до 520 КБ SRAM);

– невисока тактова частота, порівняно з настільними ПК;

– енергоспоживання, особливо у випадках автономного живлення;

– наявність/відсутність апаратної підтримки шифрування (наприклад, деякі моделі ESP32 мають апаратний AES-шифратор) [37].

Зважаючи на вищезгадане, вибір криптографічного алгоритму повинен ґрунтуватися на компромісі між рівнем захисту та ефективністю його реалізації. Порівняльна характеристика алгоритмів наведена в (табл. 1.5) [38].

Таблиця 1.5 - Порівняльна характеристика криптографічних алгоритмів

Алгоритм	Тип	Блок	Швидкість на ESP32	Примітка
AES-128	Симетричний	128 біт	Висока	Має апаратну підтримку в ESP32
RSA-2048	Асиметричний	112 біт	Низька	Великі ключі, повільна робота
ECC-256	Асиметричний	128 біт	Середня	Добре підходить для обміну ключами
ChaCha20	Симетричний	256 біт	Висока	Популярний для мобільних систем

Згідно з (табл. 1.5), асиметричні алгоритми, такі як RSA та ECC, є надто повільними для шифрування повідомлень на мікроконтролерах. Із швидких симетричних варіантів перевага надається AES-128, оскільки його апаратна підтримка в ESP32 забезпечує максимальну продуктивність при мінімальному навантаженні на процесор.

В межах теми даної роботи, доцільно обрати алгоритм AES-128 у режимі ECB/CBC як основний криптографічний механізм для шифрування переданих повідомлень. Такий вибір обумовлений наступними факторами [39]:

– баланс між продуктивністю та безпекою;

– наявність оптимізованих бібліотек для ESP32;

– можливість реалізації апаратної підтримки (в залежності від версії чіпа);

– достатній рівень захисту для сценаріїв конфіденційної комунікації при коректному управлінні ключами.

У випадках, коли є потреба в асиметричному обміні ключами, можливе використання ECC у вигляді окремого попереднього етапу для встановлення сесійного ключа AES.

1.5. Висновки та постановка задачі

У процесі дослідження встановлено, що побудова захищених систем комунікації нерозривно пов'язана з дотриманням фундаментальних принципів інформаційної безпеки: конфіденційності, цілісності та доступності, а також

автентичності та незаперечності. Ці принципи реалізуються через використання шифрування, автентифікації, контролю цілісності та заходів з резервування, що забезпечує комплексний захист переданої інформації. Ефективна протидія вимагає комплексного підходу, що поєднує в собі надійне шифрування, багатофакторну автентифікацію та безперервний моніторинг.

Фундаментом будь-якої захищеної системи є криптографічні методи. Симетричне шифрування, як-от AES, демонструє високу ефективність для великих обсягів даних, тоді як асиметричне (RSA, ECC) незамінне для безпечного обміну ключами та цифрових підписів. Попри це, поширені програмні месенджери, що використовують наскрізне шифрування, все ж мають значні вразливості. Їхня залежність від централізованих серверів та постійного доступу до інтернету створює ризики для метаданих і знижує доступність у разі мережевих збоїв.

Саме ці обмеження програмних рішень виводять на перший план апаратні підходи до захищеної комунікації. Системи, розроблені на базі мікроконтролерів (ESP32, Arduino, Raspberry Pi), дозволяють створювати автономні та ізольовані канали зв'язку, незалежні від глобальної мережі.

Проте, варто зазначити, що комерційні апаратні пристрої, попри високий рівень безпеки, залишаються дорогими та малодоступними. Це стимулює розвиток власних, доступних апаратних рішень на мікроконтролерах.

Метою роботи є розробка апаратного месенджера для захисту ведення конфіденційних переговорів на базі мікроконтролерної платформи.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– проаналізувати сучасні апаратні та програмні рішення у сфері захищеного обміну повідомленнями;

– спроектувати архітектуру системи з підтримкою базових механізмів шифрування та забезпечити ідентифікацію учасників комунікації;

– реалізувати мобільний застосунок і провести тестування працездатності та захищеності системи.

2 РОЗРОБКА АПАРАТНО-ПРОГРАМНОЇ АРХІТЕКТУРИ СИСТЕМИ

Другий розділ присвячено проектуванню архітектури захищеного месенджера на базі мікроконтролерної платформи ESP32. Буде обґрунтовано вибір апаратної та програмної платформи, розроблено структуру взаємодії компонентів, формат обміну повідомленнями та мобільні застосунки для клієнтів і адміністратора. Описуватиметься реалізація шифрування повідомлень з використанням AES, а також механізми автентифікації клієнтів на основі одноразових паролів (TOTP) і захисту каналу зв’язку через WebSocket.

2.1 Вибір архітектури системи

Запропонована архітектура апаратного месенджера базується на використанні автономної мікроконтролерної платформи ESP32 та мобільного застосунку, що функціонує у межах локальної бездротової мережі. На відміну від централізованих програмних месенджерів, ця архітектура не залежить від публічних мереж та Інтернету, що мінімізує ризики компрометації даних. Вона поєднує локальний сервер на базі ESP32 з шифруванням AES-128 та мобільний інтерфейс для користувачів, що дозволяє забезпечити повну автономність, гнучкість налаштування та високу мобільність у закритих середовищах.

Розробка захищеної системи обміну повідомленнями вимагає чітко продуманої архітектури, яка забезпечить автономність, стійкість до атак, конфіденційність переданих даних, а також відповідність вимогам до ресурсів вбудованих пристроїв. В рамках теми даної бакалаврської роботи реалізовано архітектуру, що передбачає локальну мережу типу точка-доступ (SoftAP), у якій ESP32 виконує роль сервера, а мобільні пристрої – клієнтів.

В рамках даної бакалаврської роботи реалізовано архітектуру, що передбачає використання мікроконтролерної платформи ESP32 в ролі сервера, а мобільних застосунків — у ролі клієнтів. Загальну структурну схему цієї системи наочно продемонстровано на (рис 2.1) [40].

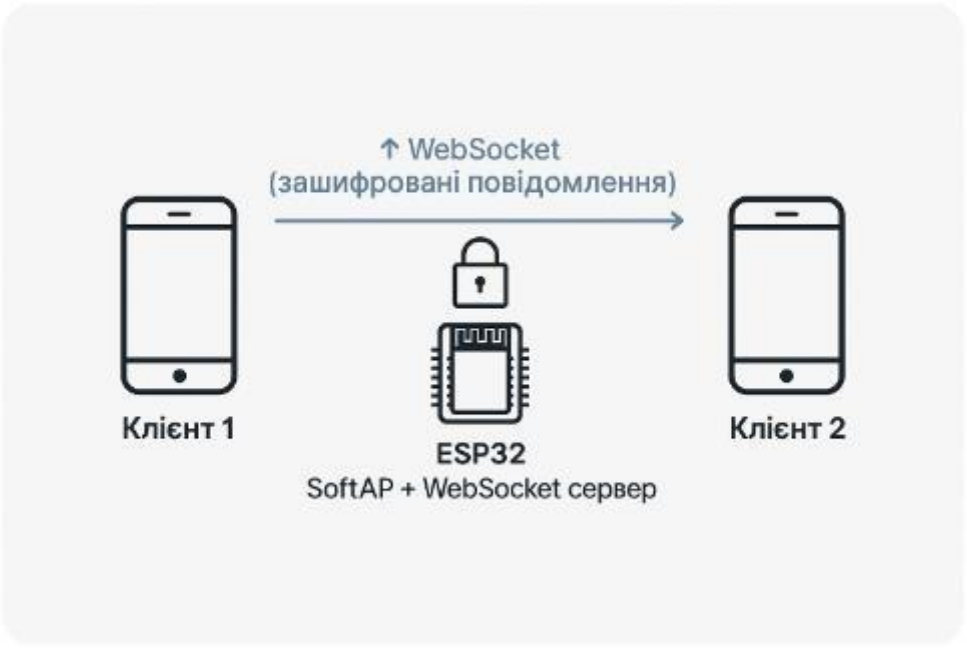


Рисунок 2.1 – Загальна структурна схема системи

Як показано на (рис. 2.1), ESP32 функціонує як центральний хаб, що створює власну Wi-Fi мережу в режимі точки доступу (SoftAP). Клієнти (мобільні пристрої) підключаються безпосередньо до цієї мережі. Увесь обмін даними між клієнтами відбувається не напряму, а через сервер на ESP32, який використовує протокол WebSocket для ретрансляції зашифрованих повідомлень. Така централізована в межах локальної мережі модель дозволяє контролювати комунікацію та застосовувати єдині правила безпеки для всіх учасників.

Система складається з таких основних компонентів:

– сервер (ESP32) – створює Wi-Fi точку доступу, запускає WebSocket-сервер, приймає та розповсюджує зашифровані повідомлення, виконує автентифікацію клієнтів;

– клієнт (мобільний додаток на Flutter) – підключається до мережі ESP32, встановлює WebSocket-з'єднання, надсилає та приймає повідомлення, шифрує та розшифровує дані.

Відповідно до запропонованої структури, логіка взаємодії компонентів наступна:

1) Ініціалізація системи. ESP32 створює SoftAP з SSID і приймає вхідні підключення.

2) Підключення клієнта. Користувач зі смартфоном підключається до Wi-Fi мережі ESP32 і запускає клієнтський додаток.

3) Установлення WebSocket-з’єднання. Установлення WebSocket-з’єднання.

4) Обмін MAC-адресою / ідентифікатором пристрою. Для уникнення дублювання повідомлень клієнт ідентифікує себе.

5) Шифрування повідомлень. Повідомлення шифрується на стороні клієнта (AES-128), надсилається на ESP32, де передається іншим користувачам без дешифрування.

6) Прийом повідомлення. Інші клієнти отримують зашифроване повідомлення і розшифровують його локально.

2.2. Обґрунтування вибору апаратної платформи

Для реалізації проєкту апаратного захищеного месенджера було обрано мікроконтролерну платформу ESP32, яка поєднує в собі достатню обчислювальну потужність, підтримку бездротових мереж, низьке енергоспоживання та компактні розміри. Вибір саме цієї платформи є зумовлений рядом технічних і функціональних переваг, що роблять її ідеальним кандидатом для створення автономного пристрою комунікації. На (див. рис. 2.2) представлено загальний вигляд плати розробника ESP32 DevKit v1, що була використана в проєкті [41].

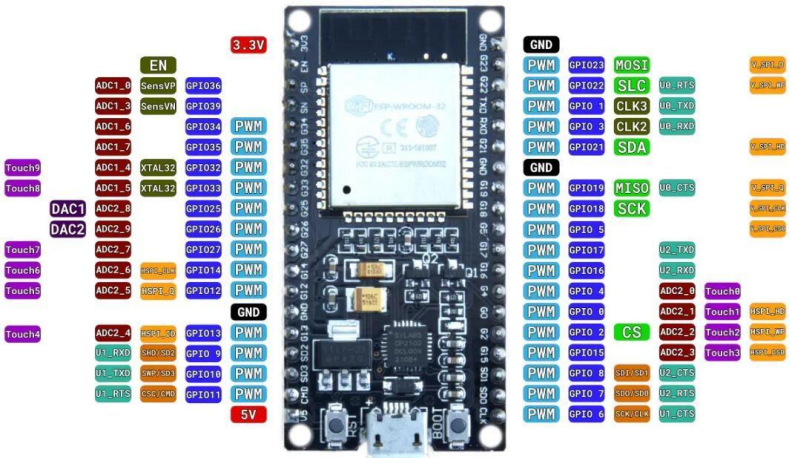


Рисунок 2.2- Мікроконтролерна платформа ESP32

Основні технічні характеристики ESP32:

– двоядерний процесор Tensilica Xtensa LX6 з частотою до 240 МГц;

– вбудовані модулі Wi-Fi та Bluetooth (BLE);

– до 520 КБ SRAM та підтримка зовнішньої пам’яті;

– розширені можливості шифрування (апаратна підтримка AES, SHA-2, RSA, ECC);

– наявність GPIO, I2C, SPI, UART, ADC, DAC для підключення периферійних пристроїв;

– низьке енергоспоживання, можливість роботи на батарейному живленні.

ESP32 здатний забезпечити не лише обробку текстових повідомлень, але й виконання шифрування/дешифрування на пристрої без потреби у додаткових модулях, що є критичним для гарантування безпеки передачі даних. Для підтвердження доцільності вибору ESP32, порівняємо її з альтернативними популярними платформами для прототипування у (табл. 2.1) [42].

Таблиця 2.1 – Порівняльна характеристика мікроконтролерних платформ

Платформа	Wi-Fi	Потужність	Безпека	Вартість	Гнучкість	Коментар
ESP32	+	Висока	+++	$3–5	Висока	Найкращий баланс можливостей
Arduino UNO	-	Низька	-	$3–7	Середня	Потребує зовнішнього Wi-Fi-модуля
Raspberry Pi	+	Дуже висока	++	$35–60	Висока	Потужна, але надмірна для задачі
STM32	+\-	Висока	+	$5–10	Висока	Складніша в розгортанні

Як видно з (табл. 2.1), ESP32 забезпечує найкраще співвідношення між потужністю, ціною та можливістю забезпечення інформаційної безпеки. На відміну від Arduino, яка не має вбудованого Wi-Fi і має обмежену продуктивність, ESP32 дозволяє реалізувати повноцінний обмін повідомленнями по захищеному каналу

без зовнішніх модулів. Raspberry Pi, хоч і потужніша, є енергозатратною, вимагає повноцінної ОС, а її ціна в кілька разів перевищує вартість ESP32.

Використання мікроконтролерної платформи ESP32 має ряд переваг, які полягають у:

– інтеграції бездротових інтерфейсів без потреби у зовнішніх компонентах;

– підтримці апаратного шифрування, що є ключовим для месенджера;

– низькій вартості, що дозволить масштабування рішення;

– великій спільноті розробників, що надають підтримку документацій та бібліотек;

– простоті у використанні в середовищі Arduino IDE або PlatformIO.

Таким чином, ESP32 є оптимальним рішенням для створення апаратного засобу конфіденційного зв’язку з обмеженими ресурсами, який працює автономно в локальній мережі.

2.3. Розробка структури взаємодії компонентів системи

У межах теми даної бакалаврської роботи було розроблено архітектуру, що дозволяє забезпечити локальний обмін повідомленнями між користувачами з використанням ESP32 як основного вузла зв’язку. Основною метою є забезпечення безпечної та конфіденційної передачі даних без залучення зовнішніх серверів чи інтернету.

Основні компоненти системи:

– ESP32 (сервер) – створює власну Wi-Fi-мережу в режимі SoftAP, приймає з’єднання від клієнтів, обробляє повідомлення, здійснює шифрування та дешифрування;

– мобільний застосунок (клієнт) – з'єднується з ESP32 через Wi-Fi, надає інтерфейс для введення/отримання повідомлень, передає унікальний ідентифікатор пристрою;

– модуль WebSocket – забезпечує двосторонню асинхронну передачу повідомлень між ESP32 та мобільним клієнтом;

– модуль шифрування AES-128 – відповідає за симетричне шифрування даних до моменту передачі через WebSocket;

– схема авторизації клієнтів – дозволяє ідентифікувати пристрої за MAC-адресами та регулювати доступ до системи;

– буфер обміну повідомленнями – тимчасово зберігає активні повідомлення у пам’яті ESP32 (історія не зберігається після сесії).

Для деталізації взаємозв'язків між ключовими функціональними блоками системи було розроблено відповідну схему, що наведена на (рис 2.3).

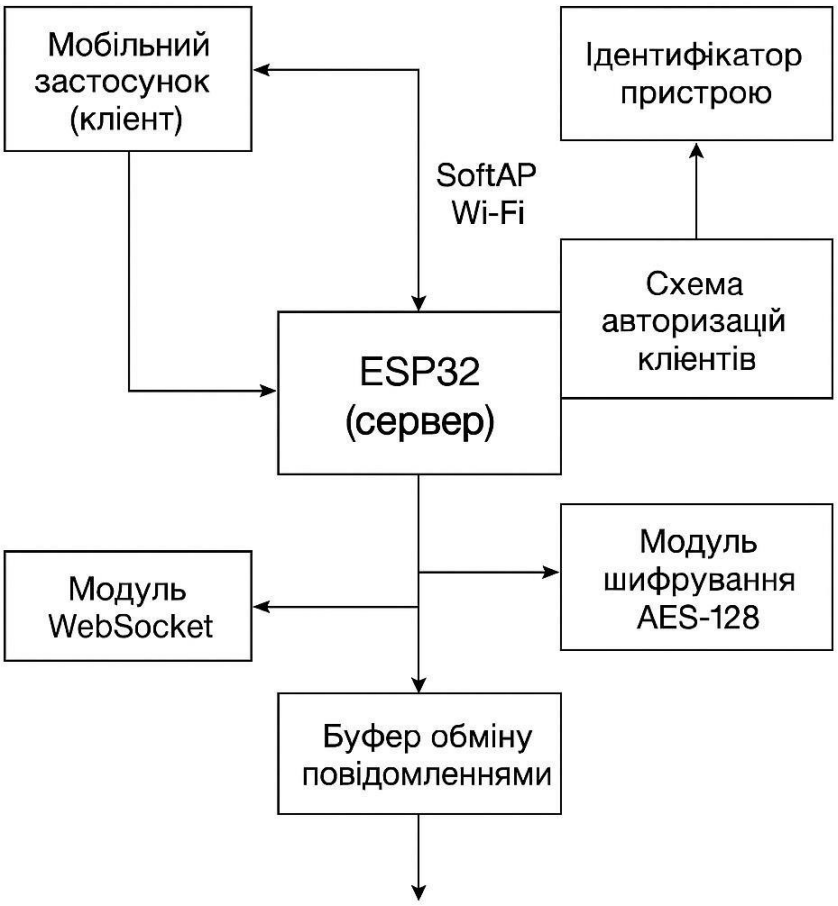


Рисунок 2.3 – Схема взаємодії компонентів системи

Наведена схема ілюструє логіку роботи системи. Мобільний застосунок (клієнт) взаємодіє з сервером на базі ESP32 через Wi-Fi з'єднання в режимі SoftAP. Для обміну даними в реальному часі використовується модуль WebSocket. Процес авторизації клієнтів відбувається на основі унікальних ідентифікаторів пристроїв. За безпеку відповідає модуль шифрування AES-128, а всі повідомлення тимчасово

обробляються через буфер обміну в пам'яті ESP32, не зберігаючись після завершення сесії.

Принцип взаємодії компонентів наступний:

1. При запуске ESP32 створює Wi-Fi-мережу (наприклад, ESP32_AP).
2. Клієнтський пристрій підключається до мережі та ініціює WebSocket-з’єднання.
3. Клієнт надсилає унікальний ідентифікатор (MAC або інший) для авторизації.
4. Після авторизації клієнт може надсилати повідомлення.
5. Повідомлення шифрується на стороні клієнта та надсилається на ESP32.
6. ESP32 дешифрує повідомлення, ідентифікує одержувача та пересилає його у зашифрованому вигляді.
7. Клієнт одержувача дешифрує повідомлення та відображає його в інтерфейсі.

Відповідно до компонентів системи і їх взаємодії, виділяються такі безпекові аспекти та реалізаційні особливості як:

– передача даних здійснюється виключно у зашифрованому вигляді (AES-128);

– дані не зберігаються на сервері після завершення сесії;

– відсутність зв’язку з інтернетом унеможливлює перехоплення сторонніми особами;

– впроваджено ручне введення секретного ключа для шифрування, що виключає централізоване управління;

– усі повідомлення мають обмеження по розміру для зменшення навантаження;

– клієнти ідентифікуються;

– клієнтська частина реалізована у Flutter із простим інтерфейсом чату.

2.4 Розробка формату обміну повідомленнями

У контексті побудови захищеної комунікаційної системи з використанням ESP32 важливо забезпечити не лише надійність та безпеку з’єднання, але й

ефективний формат обміну повідомленнями, який буде простим у реалізації, обробці й сумісним з обмеженими апаратними ресурсами.

Вимоги які висуваються до формату:

– легкість парсингу: формат повинен бути простим для розбору як на мікроконтролері, так і в мобільному застосунку;

– мінімізація розміру: короткі повідомлення зменшують навантаження на канал зв'язку;

– однозначність: уникнення неоднозначностей при інтерпретації структури;

– гнучкість: можливість розширення формату без порушення сумісності;

– безпека: формат повинен дозволяти зберігати цілісність даних після шифрування.

Під час розробки архітектури системи було розглянуто кілька поширених форматів передачі структурованих даних (табл. 2.2) [43].

Таблиця 2.2 – Порівняльна характеристика форматів обміну повідомленнями

Формат	Переваги	Недоліки
JSON	- Читабельність - Стандартизований - Широка підтримка	- Відносно великий об'єм - Потребує парсера - Не надто ефективний для мікроконтролерів
XML	- Стандартизований - Гнучка структура	- Дуже об'ємний - Важкий для обробки на обмежених пристроях
Protobuf	- Висока ефективність - Компактність - Підтримка багатьох мов	- Складність інтеграції - Неочевидний формат - Потребує генерації схем і бібліотек
CBOR / MessagePack	- Компактність - Двійковий формат	- Потребує додаткових бібліотек - Менш зручний для налагодження

Як видно з (табл. 2.2), популярні текстові формати, такі як JSON та XML, є надлишковими для мікроконтролерів через великий обсяг службових символів та потребу в ресурсомістких парсерах. Ефективні бінарні формати, як-от Protobuf, хоч і компактні, але значно ускладнюють процес розробки та налагодження.

З огляду на це, для мінімізації навантаження на ESP32 було прийнято рішення відмовитися від стандартних форматів на користь власного легковагового протоколу. Було розроблено простий формат повідомлень, що використовує розділювач (|) для відокремлення полів. Загальний вигляд формату наступний <ID>|<TYPE>|<TIMESTAMP>|<DATA>. Де:

– <ID> – унікальний ідентифікатор клієнта (наприклад, MAC-адреса або ID, згенерований застосунком);

– <TYPE> – тип повідомлення (текстове, системне, підтвердження тощо);

– <TIMESTAMP> – позначка часу у форматі UNIX time або часовому ISO-форматі;

– <DATA> – вміст повідомлення (передається в зашифрованому вигляді).

Переваги використання такого підходу:

– простота реалізації на ESP32 і Flutter;

– легка адаптація під WebSocket-протокол;

– можливість обробки без додаткових бібліотек;

– можливість перевірки цілісності (через хешування або контрольну суму при потребі).

На стороні ESP32 повідомлення обробляються за допомогою регулярного розбору по символу |, після чого виконується перевірка типу повідомлення та відповідна дія (відображення, збереження, ретрансляція або відповідь). Аналогічно й на боці мобільного застосунку.

2.5 Розробка мобільного застосунку

У межах реалізації системи захищеної комунікації було прийнято рішення розробити мобільний клієнтський застосунок, який дозволятиме користувачеві підключатися до локальної мережі ESP32, надсилати та отримувати повідомлення, а також виконувати аутентифікацію.

У якості фреймворку для створення кросплатформеного застосунку обрано Flutter – сучасне середовище, яке дозволяє створювати продуктивні та візуально

привабливі застосунки для Android та iOS на основі єдиного коду. Flutter забезпечує [44]:

– швидку розробку з підтримкою «гарячого перезапуску»;
– потужну підтримку інтерфейсів;
– великий набір бібліотек, зокрема для роботи з WebSocket, кодуванням тощо.

Основні функції мобільного клієнта:

– підключення до Wi-Fi мережі ESP32 (SoftAP);
– надсилання MAC-адреси для ідентифікації;
– введення та відображення повідомлень у чаті;
– надсилання повідомлень через WebSocket;
– відображення повідомлень з іменем відправника;
– локальна генерація OTP за TOTP-алгоритмом (опціонально);
– шифрування outgoing-повідомлень AES-128.
– Головний екран застосунку умовно буде поділено на такі компоненти:
– поле виводу повідомлень (чат);
– інтерактивне поле для введення тексту;
– кнопка «Надіслати»;
– системні повідомлення (наприклад, «Ви підключились до чату»);
– статусне повідомлення про підключення.

Взаємодія клієнта з сервером (ESP32) відбувається через протокол WebSocket. Одразу після встановлення з’єднання застосунок надсилає унікальний ідентифікатор пристрою (MAC-адресу або її хеш), після чого починається двосторонній обмін повідомленнями у режимі реального часу. Основні етапи цього процесу на стороні клієнта зображено на блок-схемі на (рис. 2.4).

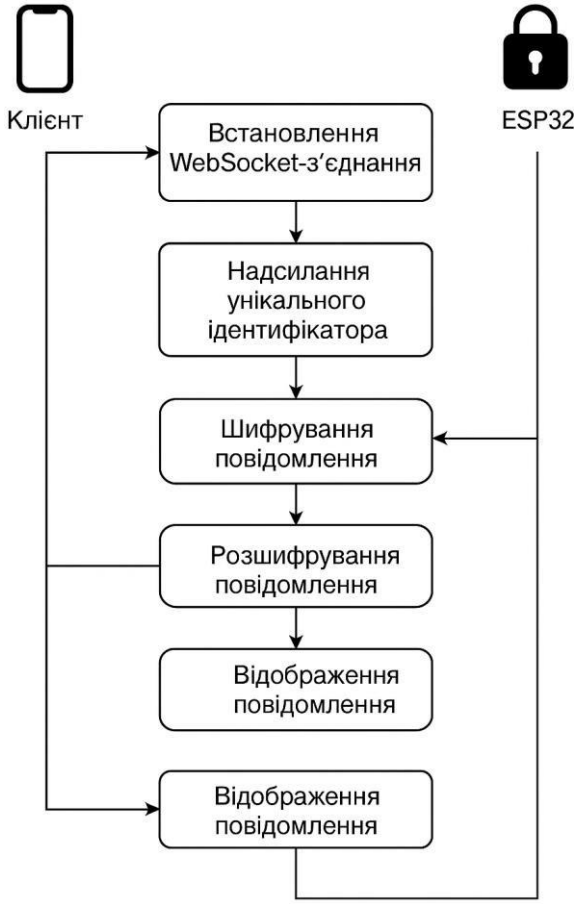


Рисунок 2.4 – Блок-схема розроблюваного мобільного застосунку

Дана блок-схема ілюструє життєвий цикл обробки повідомлення у мобільному застосунку. Процес починається зі встановлення WebSocket-з'єднання з сервером на ESP32. Після успішного підключення клієнт надсилає свій унікальний ідентифікатор для реєстрації в системі. Перед відправкою повідомлення, введеного користувачем, застосунок виконує його шифрування. Вхідні повідомлення від інших клієнтів, у свою чергу, розшифровуються локально, після чого відображаються в інтерфейсі чату.

Для цього використовуються пакети:

– web_socket_channel – для відкриття з'єднання;

– device_info_plus – для отримання унікального ID пристрою.

Кожен клієнт надсилає свій ідентифікатор, який сервер ESP32 асоціює з певним іменем у чаті (наприклад, «Клієнт 1»). Надалі кожне повідомлення

позначається іменем користувача. Повторна ідентифікація на новій сесії може відбуватися автоматично.

У наступних етапах буде реалізована функція шифрування outgoing-повідомлень на клієнті з використанням AES-128 у режимі ECB, що узгоджується з реалізацією на стороні ESP32. Ключ встановлюється вручну та зберігається на пристрої. Це дозволить реалізувати повноцінне наскрізне шифрування в межах локальної мережі.

У процесі проектування системи було прийнято рішення про створення окремого адміністративного мобільного застосунку, який дозволить оператору системи виконувати такі завдання:

– керування підключеннями користувачів (ідентифікація, відключення небажаних клієнтів);

– оновлення криптографічних ключів та секретів (AES-ключі, OTP-секрети);

– моніторинг поточного стану мережі (список підключених пристроїв, журнали подій);

– керування конфігурацією SoftAP (SSID, пароль доступу).

Даний застосунок виконуватиме функції оператора (адміністратора) системи і забезпечуватиме гнучке управління без потреби фізичного підключення до ESP32.

Реалізація функціональності адміністративного додатку розглядатиметься у Розділі 3 даної бакалаврської кваліфікаційної роботи, де висвітлюватимуться етапи практичної імплементації.

2.6 Забезпечення безпеки обміну даними

Захист переданих повідомлень є одним із ключових аспектів розробки захищеної системи комунікації. У межах даної розробки застосовано комбінацію технічних та програмних засобів для забезпечення конфіденційності, цілісності та автентичності даних.

Перед розробкою механізмів захисту було сформульовано такі вимоги до системи:

– конфіденційність – недопущення доступу сторонніх осіб до вмісту повідомлень;

– цілісність – запобігання модифікації повідомлень під час передачі;

– автентичність – перевірка джерела повідомлення (ідентифікація клієнта);

– мінімальні обчислювальні витрати – у зв’язку з обмеженими ресурсами мікроконтролерної платформи ESP32;

Для шифрування повідомлень обрано AES-128 у режимі ECB. Такий вибір обґрунтований [45]:

– високою швидкістю шифрування;

– широкою підтримкою в бібліотеках;

– достатнім рівнем безпеки для коротких повідомлень за умови дотримання обмежень режиму ECB.

Хоч режим ECB не рекомендується для шифрування великих обсягів або шаблонних даних, у нашому випадку, формат повідомлень фіксований, а сеанс не зберігає історії – отже, ризики мінімізуються.

Секретний ключ AES-128 попередньо генерується на смартфоні або окремим застосунком і передається до ESP32 вручну або через фізичне підключення на етапі ініціалізації. Такий підхід забезпечує:

– відсутність необхідності у складному обміні ключами;

– уникнення передачі ключа незахищеним каналом.

Для автентифікації користувачів використовується механізм одноразових паролів (OTP), сформованих за алгоритмом TOTP (Time-based One-Time Password). Відповідно до цього:

– кожен клієнт має спільний секрет із сервером;

– Код OTP змінюється щохвилини;

– ESP32 перевіряє код із допустимим вікном ±30 секунд;

Цей механізм дозволяє:

– підтвердити справжність клієнта;

– уникнути збереження паролів;

– не залежати від мережі Інтернет (алгоритм offline).

Взаємодія з TOTP реалізується за аналогією з Google Authenticator або FreeOTP.

З’єднання між ESP32 і мобільним застосунком встановлюється через власну Wi-Fi мережу в режимі SoftAP, без виходу в Інтернет. Передача повідомлень здійснюється за допомогою протоколу WebSocket, що дозволяє зберігати постійне з’єднання та зменшує ризики атак типу "людина посередині" (MITM).

SoftAP (англ. Software Access Point) – це режим роботи Wi-Fi-модуля, в якому мікроконтролер (наприклад, ESP32) виконує роль точки доступу (Access Point), дозволяючи іншим пристроям під’єднуватись до нього напряму, без участі зовнішнього маршрутизатора. Такий підхід дає змогу створити локальну бездротову мережу "точка-точка", що ідеально підходить для створення автономних і захищених систем комунікації [46].

Переваги використання SoftAP:

– відсутність залежності від зовнішнього інтернету або інфраструктури;
– контроль над усіма підключеними клієнтами;
– можливість реалізації безпечної ізольованої мережі.

WebSocket – це протокол зв’язку, який забезпечує двосторонню, постійну комунікацію між клієнтом (смартфоном) і сервером (ESP32). На відміну від HTTP, який є запит-відповідь, WebSocket створює постійне з'єднання, що дозволяє обмінюватися повідомленнями в режимі реального часу [47].

Переваги WebSocket у даній системі:

– мінімальні затримки при передачі повідомлень;
– підтримка повнодуплексного зв’язку;
– низьке навантаження на мережу порівняно з HTTP-запитами;

Відповідно до всього вище зазначеного загальна схема система обміну повідомленнями матиме вигляд (рис.2.5):

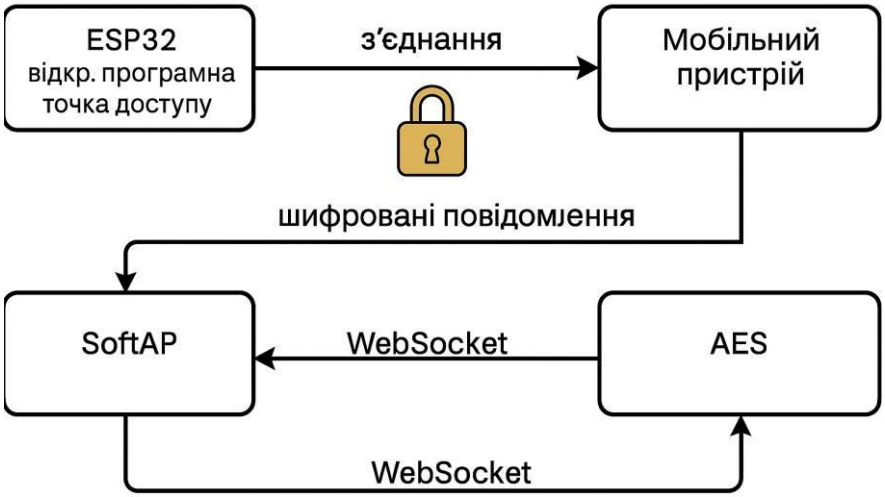


Рисунок 2.5 – Функціональна схема системи обміну повідомленнями

Відповідно до схеми зображеної на (рис.2.5) функціонування системи обміну повідомленнями відбуватиметься наступним чином:

1. Створення локальної мережі. ESP32 вмикає режим SoftAP і транслює Wi-Fi мережу з унікальним SSID. Смартфон користувача підключається до цієї мережі напряму.

2. Встановлення WebSocket-з'єднання. Після підключення до Wi-Fi клієнт (мобільний застосунок) ініціює WebSocket-з'єднання з ESP32, який виступає сервером.

3. Обмін службовими даними. Під час підключення клієнт передає унікальний ідентифікатор пристрою (наприклад, MAC-адресу). ESP32 реєструє клієнта у сесії.

4. Обмін повідомленнями. Клієнт надсилає повідомлення у заздалегідь визначеному форматі. ESP32 приймає повідомлення, дешифрує його (якщо увімкнене шифрування), зберігає в буфер і транслює всім підключеним клієнтам, окрім відправника.

5. Закінчення сесії. При відключенні клієнта ESP32 очищає тимчасову історію повідомлень (залежно від обраного режиму конфіденційності) і завершує сесію для відповідного клієнта.

2.7 Алгоритми безпеки

У сучасних умовах забезпечення конфіденційності та автентичності повідомлень у системах обміну інформацією є критично важливим завданням. У межах проекту реалізовано два основних механізми криптографічного захисту: симетричне шифрування алгоритмом AES-128 та перевірку одноразового пароля (OTP), згенерованого за алгоритмом TOTP.

Після аналізу популярних симетричних алгоритмів для вбудованих систем було прийнято рішення використовувати AES-128, як такий, що забезпечує найкращий баланс між безпекою та обчислювальною ефективністю. Для підтвердження цього вибору порівняємо найпоширеніші симетричні алгоритми у (табл. 2.3) [48].

Таблиця 2.3 - Порівняльна таблиця популярних симетричних алгоритмів

Алгоритм	Довжина ключа (біт)	Рівень безпеки	Швидкодія	Підходить для ESP32
DES	56	Низький	Висока	Так, але не рекомендується
Triple DES	112/168	Середній	Низька	Обмежено
RC4	До 2048	Низький/Сумнівний	Висока	Так, але знято з використання
AES-128	128	Високий	Висока	Так

Як демонструють дані (табл. 2.3), такі алгоритми як DES та RC4 на сьогодні не вважаються безпечними через низький або сумнівний рівень криптостійкості. Алгоритм Triple DES, у свою чергу, є занадто повільним для сучасних систем. Таким чином, AES-128 залишається єдиним варіантом із розглянутих, що поєднує високий рівень безпеки та високу швидкодію, що робить його галузевим стандартом для захищених застосунків, зокрема і на платформах з обмеженими ресурсами.

Його переваги:

– підтримка апаратного прискорення в деяких версіях ESP32;

– широко підтримується та перевірений часом;

– забезпечує достатній рівень безпеки при збереженні високої швидкодії.

Механізм працює за наступним алгоритмом (рис.2.6):

1. Генерація OTP на клієнті:

– клієнт (мобільний застосунок) зберігає спільний секретний ключ (попередньо встановлений між клієнтом і ESP32).

– Використовуючи поточний час пристрою та цей секрет, клієнт генерує OTP за допомогою TOTP-алгоритму.

– OTP додається до кожного повідомлення.

2. Перевірка OTP на ESP32:

– при отриманні повідомлення ESP32 використовує той самий секретний ключ і поточний час для генерації очікуваного OTP;

– порівнює отриманий код із очікуваним для поточного часу та для ±30 секундного вікна.

3. Обробка результату перевірки:

– якщо OTP співпадає – повідомлення приймається до обробки і пересилається іншим клієнтам;

– якщо OTP не співпадає – повідомлення відхиляється, і клієнту надсилається відповідь з повідомленням про помилку (OTP_INVALID).

Для підтвердження автентичності клієнтів у системі реалізовано механізм одноразових паролів (OTP) на основі TOTP (Time-based One-Time Password), що відповідає стандарту RFC 6238. Загальний алгоритм генерації та перевірки одноразового пароля зображено на блок-схемі на (рис 2.6).

Рисунок 2.6 – Блок-схема алгоритму генерування та перевірки OTP

Крок 1. Генерація OTP клієнтом. Першим кроком алгоритму є генерування одноразового паролю (OTP) клієнтом. Для цього використовується комбінація поточного часу та секретного ключа, що гарантує унікальність кожного OTP.

Крок 2. Генерація OTP на ESP32. Одночасно з клієнтом ESP32 також генерує OTP, використовуючи той самий секретний ключ і поточний час. Це дозволяє забезпечити однаковий результат, який згодом використовуватиметься для перевірки. Синхронізація часових міток є критично важливою для збігу обох кодів.

Крок 3. Формування повідомлення клієнтом. Клієнт додає до повідомлення, яке хоче надіслати, згенерований одноразовий пароль. Це повідомлення може містити додаткові дані, але OTP завжди має бути присутнім для ідентифікації.

Крок 4. Надсилання повідомлення до ESP32. Після формування повідомлення клієнт відправляє його на ESP32. На цьому етапі важливо, щоб повідомлення

передавалося у безпечному каналі, аби зменшити ризик перехоплення OTP сторонніми користувачами.

Крок 5. Перевірка OTP на ESP32. Отримавши повідомлення, ESP32 виділяє з нього OTP і порівнює зі своїм власним, згенерованим на основі того ж секрету. Якщо обидва коди збігаються, це означає, що клієнт справді авторизований і може надсилати повідомлення.

Крок 6. Обробка результату перевірки. Якщо OTP співпадає, ESP32 приймає повідомлення і позначає його як валідне. Якщо ж OTP не збігається, повідомлення відхиляється, а клієнту надсилається відповідь зі статусом OTP_INVALID, що повідомляє про помилку автентифікації.

Крок 7. Передача повідомлення іншим клієнтам. У випадку успішної автентифікації ESP32 пересилає отримане повідомлення іншим клієнтам. Це забезпечує обмін даними лише серед перевірених учасників системи, гарантуючи захищений обмін інформацією в мережі.

Реалізоване шифрування та OTP-перевірка значно підвищуватимуть стійкість системи до атак типу man-in-the-middle, replay attacks, а також дозволять виключити несанкціонований доступ до чату.

2.8 Висновки

У межах другого розділу виконано повний цикл проектування системи захищеного обміну повідомленнями, зокрема:

– розроблено архітектуру, що ґрунтується на ізольованій локальній Wi-Fi мережі (SoftAP) з ESP32 у ролі сервера;

– обґрунтовано вибір апаратної платформи ESP32, яка поєднує низьке енергоспоживання, достатню обчислювальну потужність і вбудовані мережеві модулі;

– побудовано структуру взаємодії між клієнтами та сервером через WebSocket, що забезпечує реальний час обміну повідомленнями;

– розроблено легковаговий формат повідомлень, оптимізований для роботи в обмеженому середовищі мікроконтролера;

– спроектовано логіку мобільного застосунок, що забезпечуватиме інтерфейс користувача для обміну повідомленнями.

Запропоновано механізми захисту даних, зокрема:

– шифрування повідомлень за допомогою AES-128;

– аутентифікацію користувачів за допомогою TOTP;

– ізольоване з’єднання без доступу до Інтернету.

Архітектурою системи передбачено створення адміністративного застосунку для керування підключеннями та ключами, що розширює функціональні можливості системи. У процесі проектування були також ідентифіковані потенційні вразливості, серед яких:

– можливість перехоплення трафіку в незашифрованому каналі WebSocket;

– відсутність обмежень на кількість підключень;

– залежність від статичних ключів без механізму їх оновлення.

Таким чином, розроблена концепція задовольняє визначені вимоги щодо автономності, захищеності та простоти використання, а підготовлена архітектура готова до

3 РЕАЛІЗАЦІЯ АПАРАТНОГО МЕСЕНДЖЕРА

У третьому розділі описуватиметься практична реалізацію всіх компонентів системи: налаштування SoftAP та WebSocket-сервера на ESP32, реалізація алгоритмів шифрування та OTP-автентифікації, взаємодія з мобільними застосунками. Буде проведено демонстраційне тестування системи з імітацією реальних сценаріїв: підключення, оновлення ключів, передача повідомлень, обробка помилок. Буде зафіксовано результати тестування, проведено аналіз продуктивності та сформульовано висновки щодо ефективності та надійності запропонованого рішення.

3.1 Вибір середовищ розробки та засобів реалізації

Перед безпосередньою реалізацією проекту було проведено аналіз доступних середовищ розробки для мікроконтролерів, мобільних додатків і засобів забезпечення захищеного обміну повідомленнями.

3.1.1 Вибір середовища розробки для ESP32

Мікроконтролерна платформа ESP32 підтримує кілька популярних середовищ для розробки[49-50]:

– ESP-IDF (Espressif IoT Development Framework) – офіційне середовище від розробника чипів Espressif;

– Arduino IDE з підтримкою ESP32 – спрощене середовище для швидкої розробки з широкою підтримкою спільноти.

Більш детально розглянемо їх особливості.

ESP-IDF – це потужне середовище для розробки на C/C++, яке надає повний доступ до всіх можливостей мікроконтролера:

– доступ до апаратних криптомодулів (апаратна підтримка AES, SHA);

– оптимізоване керування ресурсами (пам’ять, живлення);

– підтримка багатопотокової обробки завдань (FreeRTOS).

Переваги ESP-IDF:

– висока продуктивність і оптимізація;

– доступ до низькорівневих налаштувань;

– підтримка промислових стандартів.

Недоліки:

– складність початкового налаштування;

– високий поріг входження для новачків.

Arduino IDE забезпечує спрощений доступ до програмування ESP32, використовуючи звичний синтаксис Arduino та велику кількість бібліотек.

Переваги Arduino IDE:

– простота використання і швидкий старт;

– велика спільнота і документація;

– легка інтеграція з готовими бібліотеками (Wi-Fi, WebSocket, AES).

Недоліки:

– менша продуктивність порівняно з ESP-IDF;

– обмежений доступ до деяких апаратних можливостей.

З огляду на простоту реалізації, доступність бібліотек та швидкість розробки, для реалізації проекту на першому етапі обрано Arduino IDE. Надалі для підвищення ефективності можливий перехід на ESP-IDF.

3.1.2 Вибір середовища розробки для мобільного клієнта

Для реалізації мобільного клієнта було проаналізовано такі технології:

– Flutter (Dart);

– Android Studio (Java/Kotlin);

– React Native (JavaScript).

Flutter – це фреймворк від Google для розробки кросплатформених мобільних додатків на мові Dart [51].

Переваги Flutter:

– єдина кодова база для Android та iOS;

– багата екосистема віджетів для створення інтерфейсу;

– висока продуктивність близька до нативних додатків;

– підтримка WebSocket для реального часу;
– можливість інтеграції криптографічних бібліотек.

Недоліки:

– порівняно нова технологія;
– великі розміри додатків через вбудований рушій.

Android Studio – офіційне середовище для розробки Android-додатків [52].

Переваги Android Studio:

– повний доступ до нативних API;
– максимальна оптимізація для Android.

Недоліки:

– потреба в окремій розробці для iOS;
– вищий поріг входження.

React Native – фреймворк для кросплатформеного розроблення на JavaScript [53].

Переваги React Native:

– спільна кодова база для Android та iOS;
– велика спільнота розробників.

Недоліки:

– нижча продуктивність у порівнянні з Flutter;
– складнощі з інтеграцією нативних бібліотек для криптографії.

Зважаючи на переваги Flutter щодо продуктивності, підтримки WebSocket і кросплатформеності, для проекту обрано саме Flutter.

3.2 Розгортання SoftAP-сервера на ESP32

Однією з ключових функцій апаратного месенджера є створення локальної бездротової мережі, яка забезпечує ізольований канал обміну повідомленнями між учасниками переговорів. Для цього було обрано режим SoftAP (Software Access Point), що дозволяє мікроконтролерній платформі ESP32 виступати у ролі Wi-Fi точки доступу.

Режим SoftAP реалізований за допомогою стандартних можливостей Arduino-ESP32 SDK. На етапі налаштування сервер визначає основні параметри мережі:

– SSID (ім’я мережі) – унікальна назва точки доступу для ідентифікації користувачами;

– пароль – ключ доступу до мережі, який повинен відповідати вимогам безпеки (мінімум 8 символів).

Для забезпечення базового захисту було обрано WPA2-PSK як метод автентифікації, що є стандартом для захищених Wi-Fi мереж.

Під час ініціалізації SoftAP сервер встановлює такі параметри:

– максимальна кількість підключень – обмеження кількості одночасно підключених клієнтів для зниження ризику DoS-атак;

– статична IP-адреса сервера (наприклад, 192.168.4.1), яка використовується клієнтами для підключення через WebSocket.

Фрагмент коду для налаштування SoftAP на ESP32 представлено нижче:

```
WiFi.softAP("SecureChatAP", "StrongPassword123");
IPAddress IP = WiFi.softAPIP();
Serial.print("AP IP address: ");
Serial.println(IP);
```

Після активації SoftAP учасники переговорів можуть підключатися до створеної мережі за вказаним SSID і паролем. Надалі з’єднання використовується для обміну повідомленнями через WebSocket-сервер, реалізований на ESP32.

3.3 Реалізація WebSocket-сервера на ESP32

Протокол WebSocket дозволяє організувати постійне двостороннє з’єднання між клієнтом і сервером на основі TCP. На відміну від традиційних HTTP-запитів, які потребують відкриття нового з'єднання для кожного запиту, WebSocket підтримує постійне підключення, що ідеально підходить для реального часу обміну повідомленнями [54].

Основні переваги WebSocket:

– низькі затримки передачі даних;

– постійне з'єднання без необхідності перепідключення;

– підтримка одночасного обміну даними в обох напрямках (Full Duplex).

Для реалізації WebSocket-сервера на ESP32 було використано бібліотеку WebSocketsServer.h, яка забезпечує [56]:

– підтримку стандартного WebSocket-протоколу;

– обробку кількох підключень;

– обробку подій (підключення, відключення, отримання повідомлень).

– Код ініціалізації сервера:

```
#include <WiFi.h>
#include <WebSocketsServer.h>

WebSocketsServer webSocket = WebSocketsServer(81); // Порт 81

void setup() {
  WiFi.softAP("SecureChatAP", "StrongPassword123");
  webSocket.begin();
  webSocket.onEvent(onWebSocketEvent);
}

void loop() {
  webSocket.loop();
}
```

Для керування підключеннями та отримання повідомлень потрібно організувати обробку події. Код обробника подій:

```
void onWebSocketEvent(uint8_t client_num, WStype_t type, uint8_t * payload, size_t length) {
  switch (type) {
    case WStype_CONNECTED:
      Serial.printf("Client %u connected\n", client_num);
      break;
    case WStype_TEXT:
      Serial.printf("Received message: %s\n", payload);
      // Тут буде реалізовано обробку повідомлень і шифрування
      break;
    case WStype_DISCONNECTED:
      Serial.printf("Client %u disconnected\n", client_num);
      break;
  }
}
```

Для передачі текстових повідомлень передбачено спеціальний текстовий формат, розробка структури якого висвітлена в Розділі 2 даної бакалаврської роботи. Загальний вигляд формату повідомлень:

ID:OTP:ENCRYPTED_MESSAGE ,

де:

– ID – ідентифікатор користувача (наприклад, MAC-адреса);

– OTP – одноразовий пароль для автентифікації;

– ENCRYPTED_MESSAGE – шифроване повідомлення.

Отримані повідомлення будуть розшифровуватися на стороні ESP32 та передаватися іншим підключеним клієнтам.

3.4 Реалізація шифрування повідомлень на ESP32

Для забезпечення конфіденційності переданих повідомлень в апаратному месенджері реалізовано механізм шифрування на основі алгоритму AES-128 у режимі ECB. Такий підхід обрано з урахуванням обмежених ресурсів мікроконтролера ESP32 та потреби в забезпеченні базового рівня захисту при високій швидкодії.

AES (Advanced Encryption Standard) є одним із найбільш визнаних і рекомендованих симетричних алгоритмів шифрування, що стандартизований NIST та ДСТУ 7624:2014 [57].

Для проекту обрано 128-бітний ключ як баланс між безпекою і продуктивністю. Хоча режим ECB не забезпечує стійкість до певних атак (наприклад, аналізу повторюваних блоків), у поєднанні з короткими повідомленнями та ізольованою мережею ця схема вважається прийнятною.

У фінальній реалізації було прийнято рішення, що шифрування повідомлень відбуватиметься виключно на стороні ESP32. Це дозволяє:

– зменшити обчислювальне навантаження на клієнтські пристрої;

– централізувати контроль над процесом шифрування;

– забезпечити однорідну логіку обробки повідомлень.

Клієнти надсилають повідомлення у відкритому тексті, супроводжені одноразовим паролем (OTP). Після перевірки OTP, ESP32 шифрує повідомлення і пересилає його іншим клієнтам.

На апаратній платформі ESP32 використано бібліотеку mbedTLS, яка дозволяє виконувати AES-операції [58].

Ініціалізація шифрування відбувається наступним чином:

```
#include "mbedtls/aes.h"

mbedtls_aes_context aes;
uint8_t aes_key[16] = { /* 16 байт ключа */ };

void setupAES() {
  mbedtls_aes_init(&aes);
  mbedtls_aes_setkey_enc(&aes, aes_key, 128);
}
```

Процес обробки повідомлення наступний:

– ESP32 приймає повідомлення від клієнта у форматі: <ID>:<OTP>:<PLAINTEXT_MESSAGE>.

– Перевіряє дійсність OTP.

Якщо OTP валідний:

– виконується шифрування повідомлення;

– результат кодується в Base64;

– зашифроване повідомлення пересилається усім іншим клієнтам.

Приклад:

```
void encryptAndBroadcast(String plainText, uint8_t client_id) {
  uint8_t buffer[16];
  padMessageTo16(plainText,  buffer);
  mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT, buffer, buffer);
  String encoded = base64::encode(buffer, 16);
  broadcastToClients(encoded, client_id);
}
```

Клієнт отримує Base64-кодоване шифроване повідомлення, декодує його, і розшифровує локально тим самим попередньо відомим ключем. Цей підхід:

– спрощує реалізацію на ESP32;

– зменшує обсяг логіки на стороні клієнтів;

– дозволяє легко реалізовувати зміну ключів через адмін-додаток.

В початковому форматі ключ вбудовано безпосередньо в код ESP32. Надалі передбачено реалізацію механізму дистанційного оновлення ключів через адміністративний застосунок.

Повну версію коду реалізації апаратної частини на мікроконтролерній платформі ESP32 наведено в Додатку Б.

3.5 Реалізація перевірки одноразового пароля (OTP)

Для автентифікації користувачів у системі реалізовано механізм перевірки одноразового пароля (OTP) на основі алгоритму TOTP (Time-Based One-Time Password), що відповідає стандарту RFC 6238 [59].

Це дозволяє перевіряти справжність кожного повідомлення і запобігати підключенню неавторизованих клієнтів.

Принцип роботи TOTP:

– генерація пароля базується на спільному секреті та поточному часовому відрізку.

– пароль дійсний протягом короткого проміжку часу (наприклад, 30 або 60 секунд).

– клієнт генерує OTP локально і надсилає разом із повідомленням.

Оскільки ESP32 не має вбудованого годинника реального часу, передбачено синхронізацію часу під час запуску системи або за запитом клієнта. Для цього використовується команда від клієнта з поточним часом. Формат команди синхронізації, на основі якої ESP32 синхронізує свій час:

TIME_SYNC:<timestamp>

При отриманні повідомлення ESP32 виконує:

1. Розрахунок очікуваного OTP на основі збереженого секрету і поточного часу.

2. Порівняння отриманого OTP від клієнта з очікуваним.

Алгоритм перевірки наступний:

```
bool validateOTP(String receivedOtp) {
  String expectedOtp = generateCurrentOtp();
  return receivedOtp == expectedOtp;
}
```

Якщо OTP співпадає, повідомлення обробляється далі. Якщо не співпадає – сервер ігнорує повідомлення або відхиляє підключення (залежно від налаштувань).

Повідомлення як описано в п.3.3. Вигляд реального повідомлення, що передається наступний:

ABC123:847362:U2FsdGVkX1+...

3.6 Реалізація мобільного клієнта

Мобільний застосунок виступає інтерфейсом користувача для обміну захищеними повідомленнями через локальну мережу, створену ESP32.

Основні функції клієнта:

– підключення до мережі ESP32 (SoftAP);
– встановлення WebSocket-з’єднання з сервером ESP32;
– введення і надсилання повідомлень;
– генерація та передача OTP для автентифікації;
– отримання і відображення повідомлень.

Застосунок реалізовано з використанням Flutter, що забезпечує:

– кросплатформену підтримку Android та iOS;
– сучасний графічний інтерфейс користувача;
– інтеграцію з WebSocket для реального часу.

Архітектурою застосунку передбачено наявність таких головних компонентів:

– екран підключення і чату;
– механізм WebSocket-з'єднання;
– генерація OTP;
– AES-шифрування та розшифрування;
– зберігання ID пристрою.

Після визначення основних компонентів, перейдемо до їх безпосередньої реалізації.

Користувач підключається до Wi-Fi мережі, створеної ESP32, через стандартні налаштування смартфона. Після підключення до мережі застосунок ініціює з'єднання з ESP32 за локальною IP-адресою:

```
final String wsUrl = 'ws://192.168.4.1:81';
late WebSocketChannel channel;

void connectWebSocket() {
  channel = WebSocketChannel.connect(Uri.parse(wsUrl));
  isConnected = true;
}
```

Використовуючи спільний секрет, збережений у застосунку, клієнт генерує OTP на основі поточного часу. Для цього використовується бібліотека otp або аналогічна для Dart [60]:

```
String generateOtp(String secret) {
  final totp = OTP.generateTOTPCodeString(
    secret,
    DateTime.now().millisecondsSinceEpoch,
    interval: 60,
  );
  return totp;
}
```

Після введення тексту повідомлення:

– застосунок шифрує повідомлення за допомогою AES-128 з попередньо відомим ключем;

– кодує результат у Base64;

– формує повідомлення у розробленому форматі (п.3.3)

– надсилає його на ESP32 через WebSocket.

Шифрування повідомлень із використанням бібліотеки PointyCastle [61]:

```
Uint8List encryptMessage(String message, Uint8List key) {
  final padded = Uint8List(16)..setAll(0, utf8.encode(message.padRight(16)));
  final cipher = pc.AESFastEngine()..init(true, pc.KeyParameter(key));
  final output = Uint8List(16);
  cipher.processBlock(padded, 0, output, 0);
  return output;
}
```

Кодування у Base64 перед передачею:

```
String prepareEncryptedMessage(String plainText, Uint8List key) {
  final encrypted = encryptMessage(plainText, key);
  return base64Encode(encrypted);
}
```

Формування повідомлення і надсилання:

```
String deviceId = "ABC123"; // Отримано раніше
String otp = generateOtp(totpSecret);
String encryptedMessage = prepareEncryptedMessage(userInput, aesKey);

String formattedMessage = "$deviceId:$otp:$encryptedMessage";
sendMessage(formattedMessage);
```

Відображення отриманих повідомлень:

```
channel.stream.listen((data) {
  setState(() {
    messages.add("Server: $data");
  });
});
```

Отримані від сервера зашифровані повідомлення розшифровуються і відображаються у вигляді списку чату.

На ряду з клієнтською версією застосунка, передбачено окремий застосунок, що працює в режимі адміністратора безпеки, і який дозволяє:

– надсилати команди для синхронізації часу;

– оновлювати криптографічні ключі;

– перезавантажувати сервер ESP32 при потребі.

Інтерфейс користувача (рис.3.1) містить:

– поле введення повідомлення;

– кнопку "Підключитися / Відключитися";

– область для відображення історії чату;

– кнопки сканування та вставки конфігурації з адмін-додатку;

– індикатор стану з'єднання;

– для адміністратора (рис.3.2) – меню з командами управління.

Повна версія лістингу коду клієнтського застосунку реалізованого на Flutter представлено в Додатку В.

Загальний вигляд головного екрана клієнтського застосунку представлено на (рис. 3.1).

Рисунок 3.1 – Інтерфейс клієнтського додатку

На (рис. 3.1) зображено головний екран додатку одразу після запуску. У верхній частині екрана відображатиметься історія повідомлень, внизу розташовано поле для введення тексту та кнопка відправки. Під ним знаходяться функціональні кнопки: «Disconnect» для розриву з'єднання з сервером, «Сканувати конфігурацію» для зчитування QR-коду та «З буфера» для вставки конфігурації з буфера обміну. Інтерфейс адміністративного додатку, що містить додаткові команди управління, розглянуто далі **(**рис. 3.2**)**.

3.7 Реалізація управління ключами та конфігурацією

Адміністративний мобільний застосунок виконує функцію централізованого керування параметрами безпеки у розробленій системі. Його наявність дозволяє:

– уникнути потреби у перепрошивці ESP32 при кожній зміні ключів;

– оперативно синхронізувати час для коректної роботи TOTP;

– контролювати роботу WebSocket-сервера;

– гнучко керувати доступом до системи.

Завдяки розділенню на звичайний та адміністративний клієнт, система забезпечує різні рівні доступу та мінімізує ризики несанкціонованого втручання.

Адмін-додаток спирається на ту ж архітектуру, що й користувацький клієнт:

– підключення до SoftAP ESP32;

– встановлення WebSocket-з’єднання;

– надсилання команд у спеціальному форматі.

Однак на відміну від звичайного клієнта, адмін має окремий інтерфейс для відправлення службових команд (таб. 3.1) [62].

Таблиця 3.1 - Перелік підтримуваних адміністративних команд

Команда	Призначення
TIME_SYNC:<timestamp>	Синхронізація часу на ESP32 для коректної роботи TOTP
KEY_UPDATE:<new_key>	Оновлення AES-ключа для шифрування/дешифрування повідомлень
SECRET_UPDATE:<new_secret>	Оновлення спільного секрету для генерації OTP
RESTART_SERVER	Перезапуск WebSocket-сервера для оновлення конфігурації або скидання збоїв

Застосування цих команд (див. таб. 3.1) дозволяє гнучко керувати сеансом зв'язку та оперативно реагувати на потенційні загрози без необхідності фізичного доступу до серверного пристрою.

Щоб запобігти компрометації, у систему передбачено базову перевірку адміністративного доступу:

– адмінський додаток має окремий ідентифікатор (Admin ID), який ESP32 перевіряє перед виконанням команди;

– в подальших версіях можливе розширення до двофакторної автентифікації для адміністратора.

Розглянемо основні частини коду для Flutter які відповідають за основний функціонал адмін-додатку. Повну версію лістингу коду даного застосунку наведено в Додатку Г.

Форма відправлення ключа:

```
final TextEditingController keyController = TextEditingController();

ElevatedButton(
 onPressed: () {
  sendAdminCommand("KEY_UPDATE:${keyController.text}");
 },
 child: Text("Оновити ключ"),
)
```

Форма відправлення секрету OTP:

```
final TextEditingController secretController = TextEditingController();

ElevatedButton(
 onPressed: () {
  sendAdminCommand("SECRET_UPDATE:${secretController.text}");
 },
 child: Text("Оновити секрет OTP"),
)
```

Синхронізація часу:

```
void sendTimeSync() {
 final timestamp = DateTime.now().millisecondsSinceEpoch ~/ 1000;
 sendAdminCommand("TIME_SYNC:$timestamp");
}
```

Перезапуск сервера:

```
void restartServer() {
 sendAdminCommand("RESTART_SERVER");
}
```

На ESP32 всі адміністративні команди обробляються у центральному обробнику:

```
if (isAdmin(client_id)) {
 if (message.startsWith("KEY_UPDATE:")) {
  updateAesKey(newKeyHex);
 } else if (message.startsWith("SECRET_UPDATE:")) {
  updateOtpSecret(newSecret);
 } else if (message.startsWith("TIME_SYNC:")) {
  syncTime(timestamp);
 } else if (message == "RESTART_SERVER") {
  restartWebSocketServer();
 }
} else {
 Serial.println("Unauthorized admin command attempt");
}
```

Інтерфейс адмінського застосунку містить (рис.3.2):

– поле виводу згенерованого ключа шифрування;

– поле виводу згенерованого секрету TOTP;

– кнопку збереження згенерованої конфігурації безпеки, представленої ключем шифрування та секретом;

– кнопку яка відповідає за генерування ключа та секрету;

– кнопку копіювання згенерованої конфігурації в буфер обміну, для випадку якщо на одному смартфоні планується використання адмін-додатку та клієнтського додатку;

– кнопку передачі згенерованої конфігурації на мікроконтролерну платформу ESP32;

– кнопку віддаленого перезапуску мікроконтролерної платформи ESP32;

– QR-код, що містить в собі дані про згенеровану конфігурацію безпеки, яку потрібно зчитувати з клієнтського за допомогою сканера.

Для зручного доступу до цих функцій було розроблено окремий інтерфейс адміністратора, загальний вигляд якого представлено на (рис. 3.2).

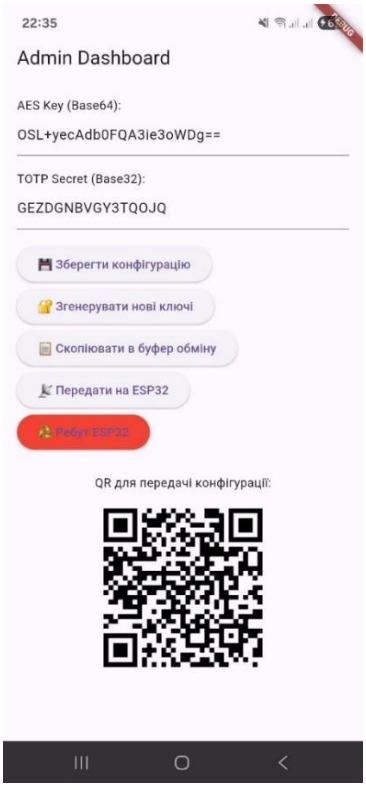


Рисунок 3.2 – Інтерфейс адмін-додатку

3.8 Тестування та перевірка функціоналу

Реалізувавши розроблені складові розробки, основним етапом є випробування роботи всього в комплексі. Основною метою тестування є перевірка працездатності всіх компонентів реалізованого апаратного месенджера, включаючи ESP32, клієнтський та адміністративний додатки. Особливу увагу зосереджено на демонстрації ключових функцій та відповідності очікуваній поведінці системи, з основним фокусом на:

– обмін повідомленнями;
– перевірку OTP;
– шифрування на ESP32;
– розшифрування на клієнтах;
– дистанційне адміністрування.

Для більшої структурованості відображення процесу тестування та демонстрації роботи системи, було прийнято рішення поділити його на умовні етапи:

1. Ініціалізація ESP32.
2. Підключення клієнтів та адміністратора.
3. Надсилання конфігураційних команд.
4. Перевірка обміну повідомленнями.
5. Віддалене перезавантаження.

Для відображення основних процесів які перетікають у системі, з боку мікроконтролерної платформи ESP32 закладено можливість їх фіксування, так зване логування. Результат логування, в силу особливостей обраного середовища, виводитиметься в так званий монітор порту. Мітки які відповідають за це відображення можна побачити в загальному коді реалізованому на ESP32 (Додаток Б). Що стосується клієнтського та адмін додатків, то відображення цих процесів можна спостерігати безпосередньо у додатках, у вигляді системних повідомлень та push-повідомлень.

Для початку роботи системи потрібно ініціалізувати мікроконтролерну платформу, оскільки вона є її основою. Результат виконання процесу ініціалізації ESP32 відображено на (рис.3.3).

Рисунок 3.3 – Ініціалізація мікроконтролерної платформи ESP32

У виводі монітора порту (рис.3.3) можна побачити інформацію щодо успішного запуску ESP32, підняття точки доступу та запуску WebSocket серверу за заданими параметрами. Також, серед виведеної інформації можна побачити поточний ключ шифрування, який встановлено в основний код за замовчуванням. Дане відображення поточного ключа потрібне буде в подальшому для демонстрації процесу його зміни.

Ініціалізувавши серверну частину, представлену ESP32, необхідно виконати зміни в конфігурації для забезпечення захищеності сеансу, а саме змінити ключ шифрування та секрет. Для цього, як йшлось в попередніх пунктах, було впроваджено адмін-додаток. Функціоналом додатку передбачений механізм передачі щойно згенерованої безпекової конфігурації по WebSocket. Процес зміни конфігурації за замовчуванням відображено на (рис.3.4).

Рисунок 3.4 – Відображення зміни конфігурації безпеки

Як видно з виводу монітора порту (рис.3.4), ESP32 зафіксувала підключення пристрою з ip адресою 192.168.4.2. Після ініціювання зміни конфігурації безпеки, в моніторі порту буде виведено повідомлення: «Конфігурацію оновлено!» та виведено інформацію про ключ шифрування та секрет TOTP.

Після успішного оновлення конфігурації на мікроконтролерній платформі, наступним етапом є під’єднання клієнтів та підготовка їх до сеансу зв’язку. Для цього, клієнтські додатки після підключення до точки доступу ESP32 повинні оновити власні конфігурації безпеки, які також представлені ключем шифрування та секретом. Наразі передбачено два варіанти оновлення конфігурації: через буфер обміну та через QR-код згенерований в адмін-додатку. Перший варіант, як зазначалось, доцільний у випадку коли на одному смартфоні встановлено і адмін-додаток, і клієнт. При тестуванні використовувалась одна ESP32 та два смартфони, один з яких поєднав функції адміністратора та клієнта. Вигляд інтерфейсів після підключення, оновлення конфігурації та авторизації клієнтського додатку на пристроях зображена на (рис.3.5-3.8).

Рисунок 3.5 – Підключення клієнта 1 до сервера

Як бачимо, в інтерфейсі додатка (рис.3.5), при під’єднанні до створеної на ESP32 точки доступу, на головному екрані виводяться системні сповіщення про результат процесу під’єднання клієнта. На основі MAC-адреси пристрою, що зчитується клієнтським додатком і обробленим на стороні ESP32, користувачам надається ID. Для продовження подальшої взаємодії з іншими користувачами, необхідно оновити конфігурацію безпеки, зазначеними раніше способами. Про результат виконання оновлення конфігурації в клієнт надійде сповіщення (рис.3.6, 3.8).

Рисунок 3.6 – Результат оновлення конфігурації безпеки через буфер обміну

Рисунок 3.7 – Інтерфейс під’єднання клієнта 2

Рисунок 3.8 – Результат успішного зчитування конфігурації через QR-код клієнтом 2

Під ’єднавшись до сервера та виконавши процедуру оновлення безпекової конфігурації, клієнти готові до обміну повідомленнями. Відображення процесу обміну повідомленнями проілюстровано на (рис. 3.9-3.10). Як бачимо, обмін повідомленнями є типовим для месенджера як такого, де відображаються власні повідомлення надіслані клієнтом та повідомлення від інших клієнтів. Так власні повідомлення відображаються у форматі: «You: текст повідомлення», а повідомлення від інших клієнтів: «Користувач (ID): текст повідомлення».

В поточному варіанті реалізації до кожного повідомлення не додавались часові мітки, оскільки даною розробкою не передбачено збереження історії повідомлень. Повідомлення зберігаються в межах встановленого сеансу.

Встановлений сеанс триватиме для окремого клієнта до тих пір, поки він не відключиться. Відключення від поточного сенсу може відбуватися двома шляхами:

через кнопку «Disconnect» або виходом з додатку. В разі збоїв підключення, або ж за інших причин можливо виконати пере підключення, кнопка «Disconnect» має другий статус «Connect». Якщо виходу з додатку не відбувалось при збої або випадковому відключені, попередні повідомлення будуть відображатись. Інші клієнти які ще залишаються у встановленому сеансі бачитимуть системне повідомлення про те, що певний користувач вийшов (рис.3.11).

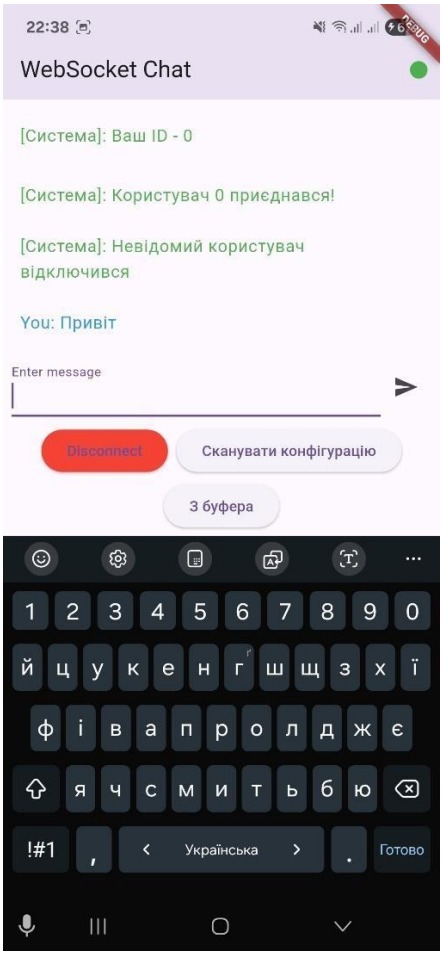


Рисунок 3.9 – Інтерфейс клієнтського додатка 1 при відправці повідомлення

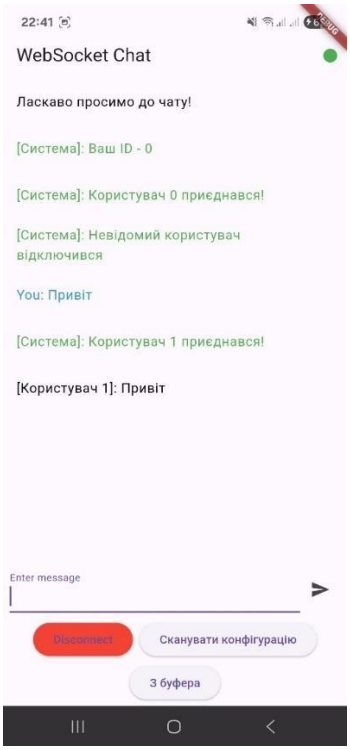


Рисунок 3.10 – Інтерфейс клієнтського додатка 1 при відправці та прийомі повідомлення від клієнтського додатка 2

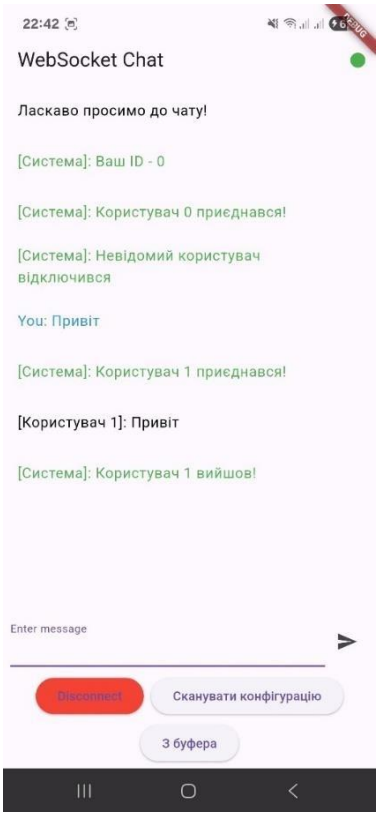


Рисунок 3.11 – Відображення процесу відключення клієнта з поточного чату

Перебіг процесу підключення клієнтів, відключення клієнтів та обміну ними повідомленнями, також чітко фіксується у моніторі порту з боку мікроконтролерної платформи ESP32 (рис.3.12).

Рисунок 3.12 – Вивід монітору порту ESP32 при підключенні, відключені і обміну повідомленнями

Описані вище процеси і результати справедливі для випадку успішного з’єднання, авторизації та оновлення конфігурації безпеки яка задається адміністратором безпеки централізовано через розроблений адмін-додаток. В разі якщо з’єднання або клієнтський додаток були якимось чином скомпрометовані, передбачена зміна конфігурації безпеки, яка доступна виключно довіреним особам.

Щоб продемонструвати ефективність механізму шифрування, було проведено тестування за сценарієм, коли один з клієнтів не пройшов процедуру оновлення конфігурації безпеки. Результат цього тестування показано на (рис. 3.13).

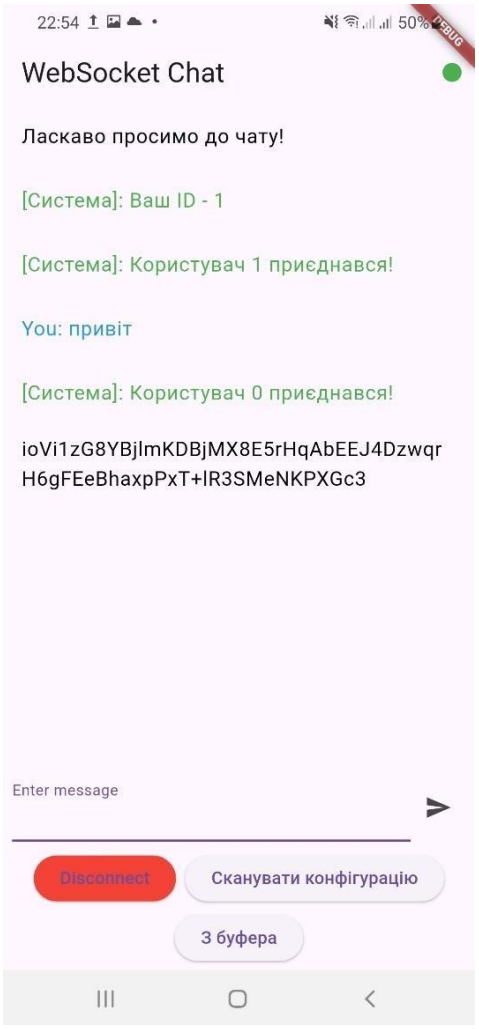


Рисунок 3.13 – Інтерфейс клієнтського додатка 2 без проходження процедури оновлення конфігурації безпеки

Як видно з (рис.3.13), клієнт який не оновив конфігурацію безпеки може відправляти повідомлення які будуть видимим тільки йому. Повідомлення які надходять від інших клієнтів будуть просто набором випадкових символів.

Наступним етапом перевірки роботи системи є перевірка реалізації віддаленого перезавантаження мікроконтролерної платформи ESP32 з адмін-додатку. Результат виконання перезавантаження фіксувався безпосередньо в моніторі порту (рис.3.14).

Рисунок 3.14 – Результат виконання віддаленого перезавантаження ESP32

Відповідно до проведеного тестування та перевірки функціоналу, узагальнимо отримані результати:

– функціонал реалізовано відповідно до вимог;

– централізоване шифрування на ESP32 працює надійно;

– OTP перевіряється з точністю до секунди;

– адмін-команди застосовуються в реальному часі.

Під час демонстраційного тестування було зафіксовано орієнтовні часові характеристики роботи ключових компонентів системи:

– обробка повідомлення на ESP32 (OTP + AES + відправлення): 10–15 мс;

– генерація OTP на клієнті: 2–3 мс;

– затримка доставки повідомлення через WebSocket у локальній мережі: не перевищує 50 мс;

– обробка адміністративних команд (оновлення ключів, синхронізація часу): до 20 мс;

– виконання перезапуску WebSocket-сервера через адмін-додаток: до 150 мс.

Отримані значення підтверджують, що система може працювати у реальному часі з мінімальними затримками, що є критично важливим для месенджера, орієнтованого на оперативну комунікацію.

Тестування працездатності розробленого програмно-апаратного комплексу проводилося методом функціональної демонстрації за ключовими сценаріями використання.

Апаратні засоби, що використовувались:

– Сервер: плата розробника ESP32 DevKitC v1.

– Клієнт 1 / Адміністратор: смартфон [Ваша модель, напр., Samsung Galaxy A52] (Android 12).

– Клієнт 2: смартфон [Ваша модель, напр., Xiaomi Redmi Note 10] (Android 11).

Програмні засоби:

– Середовище розробки для ESP32: Arduino IDE 2.3.2.

– Моніторинг роботи сервера: Serial Monitor, вбудований в Arduino IDE.

– Клієнтські та адміністративний застосунки, розроблені на Flutter 3.19.

Методика тестування полягала у покроковому виконанні тест-кейсів, що охоплюють основний функціонал системи. Результати кожного кроку фіксувалися за допомогою скріншотів інтерфейсів мобільних застосунків та виводу з монітора порту, які наведені далі в цьому розділі. Узагальнений протокол тестування наведено у Додатку Ж.

3.9 Висновки

У цьому розділі було детально описано процес реалізації апаратного месенджера, орієнтованого на захист конфіденційної інформації при локальному обміні повідомленнями. Реалізація охопила всі ключові етапи, від мережевого розгортання до криптографічного захисту даних та адміністрування системи. У результаті:

– реалізовано локальну бездротову мережу SoftAP з можливістю підключення кількох клієнтів;

– запущено WebSocket-сервер на ESP32, який обробляє повідомлення в реальному часі;

– реалізовано централізоване шифрування повідомлень на ESP32 за алгоритмом AES-128;

– розроблено клієнтський застосунок на Flutter для введення повідомлень і взаємодії з системою;

– створено адміністративний застосунок, який дозволяє оновлювати ключі, секрети та конфігурацію в реальному часі;

– проведено демонстраційне тестування, яке підтвердило стабільність і коректну роботу системи в умовах практичного використання.

Таким чином, реалізований прототип демонструє, що навіть на базі доступної мікроконтролерної платформи можна створити ефективний, захищений і автономний засіб для ведення конфіденційного обміну повідомленнями.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці апаратного засобу для захищеного обміну повідомленнями в умовах потенційної загрози несанкціонованого доступу до інформації, зокрема під час проведення конфіденційних переговорів. Як базову апаратну платформу було обрано мікроконтролерну платформу ESP32, що дозволило реалізувати автономну систему обміну даними без потреби у зовнішній інфраструктурі, зокрема інтернет-з’єднанні або централізованому сервері.

У першому розділі було проведено детальний аналіз проблематики захищеної цифрової комунікації. Окреслено основні принципи інформаційної безпеки — конфіденційність, цілісність, доступність, автентичність та незаперечність — та типові загрози, які постають у процесі обміну повідомленнями. Проаналізовано сучасні програмні засоби захищеної комунікації (Signal, Telegram, Session тощо), виявлено їх переваги й обмеження. Особливу увагу приділено апаратним рішенням, що функціонують незалежно від загальнодоступної інфраструктури, та проаналізовано криптографічні алгоритми, придатні для реалізації на ресурсно обмежених пристроях. За результатами аналізу сформульовано завдання створити власне рішення, яке базуватиметься на апаратній платформі та забезпечуватиме конфіденційний обмін повідомленнями в автономному середовищі.

У другому розділі здійснено проектування майбутньої системи. Обґрунтовано вибір архітектури: обрана модель передбачає функціонування одного ESP32 як точки доступу (SoftAP), до якої можуть під’єднуватись мобільні клієнти. В якості каналу обміну повідомленнями використано WebSocket, що дозволило забезпечити двосторонній обмін з низькою затримкою. Розроблено формат обміну повідомленнями, адаптований до обмежень мікроконтролерної платформи, з урахуванням ідентифікації клієнтів. Здійснено реалізацію двох мобільних застосунків — клієнтського для користувачів і адміністративного для конфігурації ключів та параметрів безпеки. Система автентифікації реалізована на базі одноразових паролів (TOTP), а конфіденційність повідомлень забезпечується за

допомогою симетричного шифрування AES-128. Розроблено алгоритми обробки запитів, верифікації коду, шифрування та обробки помилок авторизації.

У третьому розділі здійснено практичну реалізацію усіх компонентів. Налаштовано SoftAP та WebSocket-сервер на ESP32. Шифрування повідомлень реалізовано безпосередньо на ESP32 з використанням бібліотеки mbedTLS, а клієнти отримують уже зашифровані повідомлення, які дешифруються на основі отриманого ключа. Процес генерації та перевірки OTP реалізований як окремий блок з інтеграцією у загальний цикл автентифікації. Адміністративний застосунок дозволяє віддалено оновлювати ключі, переглядати підключених клієнтів, ініціювати перезапуск сервера.

Дана розробка спроектована з можливістю подальшого розвитку під більш складні і специфічні задачі та умови. Перспективи подальшого розвитку включають:

- впровадження більш надійних алгоритмів шифрування (наприклад, AES-256 або ChaCha20);
- розширення підтримки багатьох ESP32 в режимі mesh-з’єднань;
- логування подій та вивантаження журналів;
- додаткові методи автентифікації (наприклад біометрія);
- оптимізацію продуктивності та ще більше зниження енергоспоживання для використання в польових умовах.

Таким чином, розроблений апаратний месенджер забезпечує високий рівень безпеки передачі повідомлень у повністю ізольованому середовищі без потреби у зовнішньому зв’язку. Система може бути використана для забезпечення конфіденційності переговорів у військовій, дипломатичній, підприємницькій сферах або в умовах надзвичайних ситуацій.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Osaro M. & Christopher O. Confidentiality, Integrity, and Availability in Network Systems: A Review of Related Literature. *International Journal of Innovative Science and Research Technology*. 2023. 138–143p.
2. Палагнюк Д. М., Тищук Д. С., Березюк О. В. *Принципи забезпечення інформаційної безпеки*. Вінниця : ВНТУ, 2018. 107 с.
3. Потапова Д. *Принципи забезпечення інформаційної безпеки України*. Донецьк : Донецький національний університет імені Василя Стуса, 2022. 215 с.
4. ISO/IEC 27001:2022. *Information security, cybersecurity and privacy protection — Information security management systems — Requirements*. Geneva: International Organization for Standardization, 2022. 26 с.
5. Сема М. Ю. *Інформаційна безпека в комп'ютерних мережах: навч. посібник*. Львів : Львівська політехніка, 2021. 212 с.
6. Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. Pearson Education, 2023. 816 p.
7. Shostack A. *Threat Modeling: Designing for Security*. Indianapolis : John Wiley & Sons, 2014. 624 p.
8. Глинський Я. М., Сема М. Ю. *Безпека комп'ютерних мереж: навчальний посібник*. Львів : Видавництво Львівської політехніки, 2020. 220 с.
9. Kurose J. F., Ross K. W. *Computer Networking: A Top-Down Approach*. 8th ed. Pearson, 2021. 864 p.
10. Жуков О. Ю. *Криптографія: теорія і практика: навч. посіб*. Київ : КНЕУ, 2020. 372 с.
11. Marlinspike M., Perrin T. The Signal Protocol: Technical whitepaper : вебсайт. URL: https://signal.org/docs/ (дата звернення: 15.04.2025).
12. Electronic Frontier Foundation. Secure Messaging Scorecard : вебсайт. URL: https://www.eff.org/pages/secure-messaging-scorecard (дата звернення: 12.04.2025).
13. Telegram Messenger. Telegram FAQ: Secret Chats : вебсайт. URL: https://telegram.org/faq#secret-chats (дата звернення: 14.04.2025).

14. Threema GmbH. *Threema Cryptography Whitepaper*. URL: https://threema.ch/en/faq/security (дата звернення: 11.03.2025).

15. Oxen Project. Session Technical Documentation : вебсайт. URL: https://docs.session.io/ (дата звернення: 25.04.2025).

16. Matrix.org. End-to-End Encryption implementation guide : вебсайт. URL: https://matrix.org/docs/spec/ (дата звернення: 26.04.2025).

17. Biryukov A., Khovratovich D. Privacy issues and data leakage in popular messaging apps. *Journal of Information Security Research*. 2020. Vol. 11, Issue 2. P. 45–64.

18. Espressif Systems. *ESP32 Technical Reference Manual*. URL: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (дата звернення: 28.04.2025).

19. Banzi M., Shiloh M. *Getting Started with Arduino*. 4th ed. Sebastopol : O'Reilly Media, 2022. 250 p.

20. Raspberry Pi Foundation. Raspberry Pi Documentation : вебсайт. URL: https://www.raspberrypi.org/documentation/ (дата звернення: 29.04.2025).

21. Meshtastic. Open Source Mesh Communication Project : вебсайт. URL: https://meshtastic.org/ (дата звернення: 01.05.2025).

22. Pine64. PineDio Stack Documentation : вебсайт. URL: https://wiki.pine64.org/ (дата звернення: 02.05.2025).

23. GitHub. ESP32 Secure Chat Examples. URL: https://github.com/topics/esp32-chat (дата звернення: 03.05.2025).

24. Silent Circle. Blackphone Secure Smartphone : вебсайт. URL: https://www.silentcircle.com/ (дата звернення: 05.05.2025).

25. General Dynamics. Sectéra Edge Secure Mobile Device : вебсайт. URL: https://gdmissionsystems.com/ (дата звернення: 06.05.2025).

26. Bittium Corporation. Tough Mobile Solutions : вебсайт. URL: https://www.bittium.com/ (дата звернення: 07.05.2025).

27. Stallings, W. *Network Security Essentials: Applications and Standards*. 7th ed. Pearson Education, 2022. 432 p.

28. Ніконенко С. В. *Програмування мікроконтролерів: навчальний посібник*. Київ : Ліра-К, 2021. 168 с.

29. Єфімов В. П. *Програмування вбудованих систем: курс лекцій*. Київ : КПІ ім. Ігоря Сікорського, 2022. 112 с.

30. Google Developers. *Flutter Documentation* : вебсайт. URL: https://docs.flutter.dev/ (дата звернення: 10.05.2025).

31. Dart Language Tour : вебсайт. URL: https://dart.dev/guides/language/language-tour (дата звернення: 11.05.2025).

32. Dworkin M. *NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation*. NIST, 2001. 66 p.

33. Mielke D. *RFC 6238: TOTP: Time-Based One-Time Password Algorithm*. IETF, 2011. 16 с. URL: https://www.rfc-editor.org/rfc/rfc6238 (дата звернення: 19.05.2025).

34. Krawczyk H., Bellare M., Canetti R. *RFC 2104: HMAC: Keyed-Hashing for Message Authentication*. IETF, 1997. 11 с. URL: https://www.rfc-editor.org/rfc/rfc2104 (дата звернення: 20.05.2025).

35. OWASP Foundation. WebSocket Security : вебсайт. URL: https://owasp.org/www-community/attacks/WebSocket_Scanning (дата звернення: 21.05.2025).

36. Espressif Systems. *ESP-IDF Programming Guide (for ESP32)*. Version 5.x. 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/ (дата звернення: 11.05.2025).

37. Arduino Project Hub. *Arduino IDE User Guide*. Arduino.cc, 2023. URL: https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-getting-started (дата звернення: 12.05.2025).

38. *Android Developer Documentation* : вебсайт. URL: https://developer.android.com/studio (дата звернення: 13.05.2025).

39. *React Native Documentation* : вебсайт. URL: https://reactnative.dev/docs/getting-started (дата звернення: 14.05.2025).

40. *IEEE Std 802.11i-2004: Amendment to IEEE Std 802.11 for Medium Access Control (MAC) Security Enhancements*. IEEE, 2004. 190 с.

41. *RFC 6455: The WebSocket Protocol*. IETF, 2011. 71 с. URL: https://www.rfc-editor.org/rfc/rfc6455 (дата звернення: 15.05.2025).

42. Sattler M. ArduinoWebSockets: WebSocket Server and Client for Arduino. GitHub Repository. 2023. URL: https://github.com/Links2004/arduinoWebSockets (дата звернення: 16.05.2025).

43. ДСТУ 7624:2014. *Інформаційні технології. Методи криптографічного захисту. Блоковий шифр AES*. [Чинний від 2016-03-01]. Вид. офіц. Київ : ДП «УкрНДНЦ», 2015. 228 с.

44. ARM Mbed. Mbed TLS: An open source, portable, easy to use, readable and flexible SSL library : вебсайт. 2024. URL: https://www.trustedfirmware.org/projects/mbed-tls/ (дата звернення: 18.05.2025).

45. OTP - Dart Package // pub.dev. 2023. URL: https://pub.dev/packages/otp (дата звернення: 20.05.2025).

46. Pointy Castle Development Team. Pointy Castle: A Dart library for cryptographic algorithms // pub.dev. 2023. URL: https://pub.dev/packages/pointycastle (дата звернення: 21.05.2025).

47. Pressman R. S., Maxim B. R. *Software Engineering: A Practitioner's Approach*. 9th ed. McGraw-Hill Education, 2020. 896 p.

48. Myers G. J., Sandler C., Badgett T. *The Art of Software Testing*. 3rd ed. Wiley, 2011. 256 p.

49. Norman D. *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books, 2013. 368 p.

50. Rubin J., Chisnell D. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. 2nd ed. Wiley, 2008. 368 p.

51. *Про захист персональних даних : Закон України від 01.06.2010 р. № 2297-VI*. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 25.05.2025).

52. *Про основні засади забезпечення кібербезпеки України : Закон України від 05.10.2017 р. № 2163-VIII*. URL: https://zakon.rada.gov.ua/laws/show/2163-19 (дата звернення: 26.05.2025).

53. Furtak, T. *Arduino для початківців: практичний посібник*. Київ : Арій, 2021.

54. Schweighofer J. *Mastering Arduino Libraries: A practical guide to building efficient and scalable Arduino programs using libraries*. Packt Publishing, 2022. 382 p.

55. Amin N. *Flutter Complete Reference 2023: Master Cross-Platform Mobile App Development*. 2nd ed. Leanpub, 2023. 560 p.

56. Haque T. *Practical Flutter: Improve your Mobile Development with Google's Latest Open-Source SDK*. Apress, 2021. 416 p.

57. Sych D. Dart Apprentice: Learn to program with Dart. Razeware LLC, 2023. URL: https://www.kodeco.com/books/dart-apprentice (дата звернення: 13.05.2025).

58. Андрусенко В. В. Розробка мобільних додатків засобами Flutter. *Збірник наукових праць ХНУРЕ*. 2022. №4. С. 81–87.

59. Sauter, M. *From GSM to LTE-advanced Pro and 5G: An Introduction to Mobile Networks and Mobile Broadband*. Wiley, 2021. 304 p.

60. Albahari, B. *Secure IoT Communications with TLS and MQTT*. O'Reilly Media, 2021.

61. Jose, M. *Practical Flutter: Improve your Mobile Development with Google's Framework*. Apress, 2020.50p.

62. Нікітчук, Ю. І., Ткачук, М. І. *Технології захисту інформації: Навч. посібник*. Львів: Львівська політехніка, 2020. 40р.

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

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
Ю.Є. Яремчук
«20» березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської кваліфікаційної роботи на тему:
Розробка апаратного месенджера для захисту ведення конфіденційних переговорів на базі мікроконтролерної платформи

08-72.БКР.009.00.100.ТЗ

Керівник бакалаврської кваліфікаційної роботи
к.ф.-м.н., доцент каф. МБІС
Шиян А.А.
"____" _____

Вінниця – 2025р.

1. Найменування та область застосування

Апаратний месенджер на базі мікроконтролерної платформи.

Область застосування: у сферах, де критично важливим є забезпечення конфіденційності, цілісності та автентичності переданої інформації – зокрема в оборонній, урядовій, розвідувальній, корпоративній та медичній галузях.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 97 від 20.03.2025 р.

3. Мета та призначення розробки

3.1 Мета: розробка апаратного месенджера для захищеного ведення конфіденційних переговорів на базі мікроконтролерної платформи з використанням технологій локального бездротового зв’язку, базових криптографічних алгоритмів та мобільного інтерфейсу для зручної взаємодії з користувачем.

3.2 Призначення: завдяки автономній архітектурі, використанню шифрування та OTP-аутентифікації, він ефективно працює без підключення до Інтернету, що робить його придатним для використання в умовах обмеженої довіри до інфраструктури або підвищених загроз витоку інформації.

4. Джерела розробки

4.1 Osaro M. & Christopher O. Confidentiality, Integrity, and Availability in Network Systems: A Review of Related Literature. International Journal of Innovative Science and Research Technology, 2023. 20 p.

4.2 Палагнюк Д. М., Тищук Д. С., Березюк О. В. Принципи забезпечення інформаційної безпеки. Вінницький національний технічний університет. Вінниця : Видавнича група BHV, 2018. 107 с.

4.3 Потапова Д. Принципи забезпечення інформаційної безпеки України. Донецьк: Донецький національний університет імені Василя Стуса, 2022. 215с.

4.4 Сема, М. Ю. Інформаційна безпека в комп’ютерних мережах: навч. посібник / М. Ю. Сема. Львів: Львівська політехніка, 2021. 212 с.

5. Вимоги до розробки

5.1 Вимоги до функціональних характеристик:

5.1.1 Забезпечення захищеного обміну повідомленнями між клієнтськими пристроями через мікроконтролерний сервер ESP32 з використанням шифрування.

5.1.2. Підтримка аутентифікації користувачів.

5.1.3. Автономна робота без підключення до Інтернету..

5.1.4. Візуалізація повідомлень та статусів у графічному інтерфейсі мобільного застосунку, окремо для клієнта та адміністратора.

5.1.5. Віддалене керування конфігурацією безпеки (оновлення ключів, перезавантаження, блокування доступу) через окремий мобільний застосунок.

5.1.6. Обмеження доступу тільки для авторизованих користувачів із попередньо зареєстрованими MAC-адресами.

5.1.7. Збереження сеансової конфіденційності.

5.1.8. Підтримка кількох клієнтів одночасно (не менше 2-х), з динамічною обробкою повідомлень.

5.2 Вимоги до надійності:

5.2.1 Система повинна працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Програмна частина повинна виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

5.3.1. Основна апаратна платформа:

– плата ESP32 DevKit v1 або еквівалент з підтримкою Wi-Fi, мін. 240 МГц, 2 ядра;

– мінімум 512 КБ оперативної пам’яті;

– підтримка WebSocket, криптографічних бібліотек, SoftAP.

5.3.2. Живлення:

– можливість автономного живлення від акумулятора (3.7–5 В), а також через USB.

5.3.3. Мобільні клієнтські пристрої:

– смартфони або планшети з ОС Android 8.0 або вище;

– підтримка Flutter-додатку, мінімум 2 ГБ оперативної пам’яті;

– підтримка Wi-Fi-з'єднання.

5.3.4. Програмне забезпечення для розробки:

– Arduino IDE 2.x або PlatformIO;

– Flutter SDK версії 3.10 або новіші;

– Dart версії 3.0 або новіші.

5.3.5 Ввимоги до техніки безпеки при роботі з системою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

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

7. Вимоги до технічного захисту інформації

7.1 Неможливість отримання доступу сторонніх користувачів до відкритих даних.

7.2. Обробка даних без можливості перегляду їх вмісту третіми сторонами.

8. Техніко-економічні показники

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

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

9. Стадії та етапи розробки

№	Назва та зміст етапу	Термін виконання	
		початок	закінчення
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025
2	Аналіз предметної області обраної теми	24.03.2025	13.04.2025
3	Розробка, реалізація	14.04.2025	27.04.2025
4	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025
6	Виправлення, уточнення, корегування бакалаврської кваліфікаційної роботи	28.05.2025	08.06.2025
7	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

– ПЗ до бакалаврської кваліфікаційної роботи;

– демонстрація результату бакалаврської кваліфікаційної роботи;

– презентація;

– відзив керівника роботи;

– відзив рецензента.

Технічне завдання до виконання прийняв _____ Кравчук В.О

Додаток Б. Лістинг коду мікроконтролерної платформи ESP32

```
#include <WiFi.h>
#include <WebSocketsServer.h>
#include "mbedtls/aes.h"
#include <base64.h>
#include <map>
#include <vector>
#include <algorithm>

const char* ssid = "ESP32_AP";
const char* password = "12345678";

WebSocketsServer webSocket(81);

std::vector<uint8_t> clients;
std::map<String, uint8_t> macToId;
std::map<uint8_t, String> socketToMac;
uint8_t nextClientId = 0;

uint8_t aesKey[16] = {
  0x2b, 0x7e, 0x15, 0x16,
  0x28, 0xae, 0xd2, 0xa6,
  0xab, 0xf7, 0x97, 0x75,
  0x46, 0x64, 0x1f, 0x67
};

char totpSecret[32] = "JBSWY3DPEHPK3PXP";

void hexStringToBytes(const String& hexString, uint8_t* byteArray, size_t length) {
  for (size_t i = 0; i < length; i++) {
    String byteString = hexString.substring(i * 2, i * 2 + 2);
    byteArray[i] = (uint8_t) strtol(byteString.c_str(), NULL, 16);
  }
}

String encryptAES(String message) {
  mbedtls_aes_context aes;
  mbedtls_aes_init(&aes);
  mbedtls_aes_setkey_enc(&aes, aesKey, 128);

  size_t inputLength = message.length();
  size_t paddedLength = ((inputLength + 15) / 16) * 16;
  unsigned char input[paddedLength];
  unsigned char output[paddedLength];

  memset(input, 0, paddedLength);
  memcpy(input, message.c_str(), inputLength);
```

```
  for (size_t i = 0; i < paddedLength; i += 16) {
    mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT, input + i, output + i);
  }

  mbedtls_aes_free(&aes);

  return base64::encode(output, paddedLength);
}

void printAesKeyBase64() {
  String b64 = base64::encode(aesKey, 16);
  Serial.print("ESP32 AES Key (base64): ");
  Serial.println(b64);
}

void broadcastEncrypted(String message, uint8_t sender) {
  String encrypted = encryptAES(message);
  for (uint8_t client : clients) {
    if (client != sender) {
      webSocket.sendTXT(client, "Encrypted:" + encrypted);
    }
  }
}

void broadcastPlain(String message) {
  for (uint8_t client : clients) {
  webSocket.sendTXT(client, message);
  }
}

void onWebSocketEvent(uint8_t num, WStype_t type, uint8_t* payload, size_t length) {
  String message = String((char*)payload);

  // DEBUG: Друкуємо вхідне повідомлення (HEX)
  String hexMsg;
  for (size_t i = 0; i < length; i++) {
    if (payload[i] < 16) hexMsg += "0";
    hexMsg += String(payload[i], HEX);
  }
  Serial.print("Вхідне повідомлення (HEX): ");
  Serial.println(hexMsg);

  switch (type) {
    case WStype_CONNECTED: {
      IPAddress ip = webSocket.remoteIP(num);
      Serial.printf("Клієнт [%u] підключився з IP: %s\n", num, ip.toString().c_str());
      webSocket.sendTXT(num, "Ласкаво просимо до чату!");
```

```
  break;
 }

 case WStype_TEXT: {
  Serial.printf("Клієнт [%u] надіслав: %s\n", num, message.c_str());

  // --- РЕБУТ: Адмін-команда ---
  if (message == "REBOOT") {
   webSocket.sendTXT(num, "[Система]: Ребут ESP32 через адміністратора...");
   Serial.println(" Виконується перезавантаження ESP32 за запитом адміністратора.");
   delay(200);
   ESP.restart();
   return;
  }

  if (message.startsWith("SET_CONFIG:")) {
   String configData = message.substring(11);
   int sep = configData.indexOf(';');
   if (sep > 0) {
    String hexKey = configData.substring(0, sep);
    String newSecret = configData.substring(sep + 1);

    hexStringToBytes(hexKey, aesKey, 16);
    strncpy(totpSecret, newSecret.c_str(), sizeof(totpSecret) - 1);
    totpSecret[sizeof(totpSecret) - 1] = '\0';

    Serial.println("Конфігурацію  оновлено:");
    Serial.print("AES Key (Hex): "); Serial.println(hexKey);
    Serial.print("TOTP Secret: "); Serial.println(newSecret);
    printAesKeyBase64();

    webSocket.sendTXT(num, "[Система]: Конфігурацію оновлено");
   } else {
    webSocket.sendTXT(num, "[Система]: Помилка формату конфігурації");
   }
   return;
  }

  if (message.startsWith("MAC:")) {
   String mac = message.substring(4);
   if (macToId.find(mac) == macToId.end()) {
    macToId[mac] = nextClientId++;
   }

   uint8_t clientId = macToId[mac];
   socketToMac[num] = mac;

   if (std::find(clients.begin(), clients.end(), num) == clients.end()) {
```

```
            clients.push_back(num);
           }

           Serial.printf("MAC %s присвоєно ID: %u\n", mac.c_str(), clientId);

           webSocket.sendTXT(num, "[Система]: Ваш ID - " + String(clientId));
           broadcastPlain("[Система]: Користувач " + String(clientId) + " приєднався!");
          } else {
           if (socketToMac.count(num)) {
            String senderMac = socketToMac[num];
            uint8_t senderId = macToId[senderMac];
            String formatted = "[Користувач " + String(senderId) + "]: " + message;
            webSocket.sendTXT(num, "You: " + message);
            broadcastEncrypted(formatted, num);
           } else {
            webSocket.sendTXT(num, "[Система]: Невідомий користувач. Надішліть MAC-
адресу.");
           }
          }
          break;
         }

         case WStype_DISCONNECTED: {
          Serial.printf("Клієнт [%u] відключився\n", num);
          clients.erase(std::remove(clients.begin(), clients.end(), num), clients.end());

          if (socketToMac.count(num)) {
           uint8_t id = macToId[socketToMac[num]];
           broadcastPlain("[Система]: Користувач " + String(id) + " вийшов!");
           socketToMac.erase(num);
          } else {
           broadcastPlain("[Система]: Невідомий користувач відключився");
          }
          break;
         }
        }
       }

       void setup() {
        Serial.begin(115200);
        WiFi.softAP(ssid, password);
        Serial.println("Wi-Fi запущено: ESP32_AP");

        webSocket.begin();
        webSocket.onEvent(onWebSocketEvent);
        Serial.println("WebSocket-сервер працює на ws://192.168.4.1:81");
        printAesKeyBase64();
       }
```

```
void loop() {
  webSocket.loop();
}
```

Додаток В. Лістинг коду клієнтського додатку

```
import 'package:flutter/material.dart';
import 'package:web_socket_channel/web_socket_channel.dart';
import 'package:device_info_plus/device_info_plus.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:mobile_scanner/mobile_scanner.dart';
import 'dart:io';
import 'dart:convert';
import 'dart:typed_data';
import 'dart:math';
import 'package:flutter/services.dart';
import 'package:pointycastle/export.dart' as pc;
import 'package:uuid/uuid.dart';

void main() {
  runApp(MaterialApp(
    home: ChatScreen(),
  ));
}

class ChatScreen extends StatefulWidget {
  @override
  _ChatScreenState createState() => _ChatScreenState();
}

class _ChatScreenState extends State<ChatScreen> {
  final String wsUrl = 'ws://192.168.4.1:81';
  WebSocketChannel? channel;
  TextEditingController controller = TextEditingController();
  List<String> messages = [];
  bool isConnected = false;
  String? deviceId;
  Uint8List key = Uint8List(16);
  String totpSecret = '';
  int? myId;

  @override
  void initState() {
    super.initState();
    loadKeysFromStorage();
    initDeviceIdAndConnect();
  }

  Future<void> loadKeysFromStorage() async {
    final prefs = await SharedPreferences.getInstance();
    final savedKey = prefs.getString('aes_key');
    final savedSecret = prefs.getString('totp_secret');
```

```
  if (savedKey != null && savedSecret != null) {
    setState(() {
      key = base64.decode(savedKey);
      totpSecret = savedSecret;
    });
    print(" Flutter AES Key (base64): ${base64.encode(key)}");
    print(" Flutter AES Key (HEX): ${key.map((b) => b.toRadixString(16).padLeft(2, '0')).join()}");
  }
 }

 Future<void> loadConfigFromClipboard() async {
  final clipboardData = await Clipboard.getData('text/plain');
  if (clipboardData?.text != null) {
    try {
      final jsonData = jsonDecode(clipboardData!.text!);
      final decodedKey = base64.decode(jsonData['aes_key']);
      final totp = jsonData['totp_secret'];

      final prefs = await SharedPreferences.getInstance();
      await prefs.setString('aes_key', jsonData['aes_key']);
      await prefs.setString('totp_secret', totp);

      setState(() {
        key = decodedKey;
        totpSecret = totp;
      });
      print(" Flutter AES Key (base64): ${base64.encode(key)}");
      print(" Flutter AES Key (HEX): ${key.map((b) => b.toRadixString(16).padLeft(2,
'0')).join()}");

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(' Конфігурацію зчитано з буфера обміну')),
      );
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(' Помилка обробки буфера: $e')),
      );
    }
  }
 }

 Future<void> initDeviceIdAndConnect() async {
  final prefs = await SharedPreferences.getInstance();
  String? storedId = prefs.getString('custom_device_id');

  if (storedId == null) {
    String newId;
```

```
    if (Platform.isAndroid) {
      final deviceInfo = DeviceInfoPlugin();
      final androidInfo = await deviceInfo.androidInfo;
      newId = androidInfo.id ?? const Uuid().v4(); // Оновлено тут!
    } else if (Platform.isIOS) {
      final deviceInfo = DeviceInfoPlugin();
      final iosInfo = await deviceInfo.iosInfo;
      newId = iosInfo.identifierForVendor ?? const Uuid().v4();
    } else {
      newId = const Uuid().v4();
    }
    storedId = newId;
    await prefs.setString('custom_device_id', storedId);
  }
  setState(() {
    deviceId = storedId;
  });
  print('Device ID to send: $deviceId');
  connectWebSocket();
}

void connectWebSocket() {
  channel = WebSocketChannel.connect(Uri.parse(wsUrl));

  channel!.stream.listen((message) {
    _handleIncomingMessage(message);
  }, onError: (error) {
    print('WebSocket Error: $error');
  }, onDone: () {
    setState(() {
      isConnected = false;
    });
  });

  setState(() {
    isConnected = true;
  });

  Future.delayed(Duration(milliseconds: 500), () {
    if (deviceId != null) {
      channel!.sink.add("MAC:$deviceId");
      print(" Надіслано на сервер: MAC:$deviceId");
    }
  });
}

void disconnectWebSocket() {
  channel?.sink.close();
```

```
  setState(() {
    isConnected = false;
  });
}

void sendMessage() {
  if (controller.text.isNotEmpty && channel != null) {

    channel!.sink.add(controller.text);

    controller.clear();
  }
}

void _handleIncomingMessage(String message) {
  if (message.startsWith("[Система]: Ваш ID - ")) {
    final idString = message.replaceFirst("[Система]: Ваш ID - ", "");
    int? assignedId = int.tryParse(idString);
    if (assignedId != null) {
      setState(() {
        myId = assignedId;
      });
    }
    messages.add(message);
    return;
  }
  if (message.startsWith("[Система]:")) {
    setState(() {
      messages.add(message);
    });
    return;
  }
  final decrypted = decryptMessage(message);

  final userMsgPattern = RegExp(r'^\[Користувач (\d+)\]: (.+)$');
  final match = userMsgPattern.firstMatch(decrypted);
  if (match != null && myId != null) {
    final int senderId = int.tryParse(match.group(1)!) ?? -1;
    final msgText = match.group(2)!;
    if (senderId == myId) {
      return;
    } else {
      setState(() {
        messages.add("[Користувач $senderId]: $msgText");
      });
      return;
    }
  }
```

```
        setState(() {
          messages.add(decrypted);
        });
      }

      String encryptMessage(String plainText) {
        final padded = _padToBlockSize(utf8.encode(plainText));
        final cipher = pc.ECBBlockCipher(pc.AESFastEngine())
          ..init(true, pc.KeyParameter(key));
        final encrypted = _processBlocks(cipher, padded);
        return base64.encode(encrypted);
      }

      String decryptMessage(String base64Message) {
        try {
          if (base64Message.startsWith("Encrypted:")) {
            base64Message = base64Message.replaceFirst("Encrypted:", "");
          }
          final encrypted = base64.decode(base64Message);
          print(" Decrypt input (base64): $base64Message");
          print(" Decrypt input (HEX): ${encrypted.map((b) => b.toRadixString(16).padLeft(2,
'0')).join()}");
          final cipher = pc.ECBBlockCipher(pc.AESFastEngine())
            ..init(false, pc.KeyParameter(key));
          final decrypted = _processBlocks(cipher, encrypted);
          return utf8.decode(decrypted.where((b) => b != 0).toList());
        } catch (_) {
          return base64Message;
        }
      }

      List<int> _padToBlockSize(List<int> data) {
        const blockSize = 16;
        int padLength = blockSize - (data.length % blockSize);
        return data + List.filled(padLength, 0);
      }

      List<int> _processBlocks(pc.BlockCipher cipher, List<int> input) {
        final output = <int>[];
        for (var offset = 0; offset < input.length; offset += cipher.blockSize) {
          final block = Uint8List.fromList(input.sublist(offset, offset + cipher.blockSize));
          output.addAll(cipher.process(block));
        }
        return output;
      }

      void scanQRConfiguration() async {
```

```
            final result = await Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => QRScannerScreen()),
            );

            if (result != null && result is String) {
              try {
                final jsonData = jsonDecode(result);
                final decodedKey = base64.decode(jsonData['aes_key']);
                final totp = jsonData['totp_secret'];

                final prefs = await SharedPreferences.getInstance();
                await prefs.setString('aes_key', jsonData['aes_key']);
                await prefs.setString('totp_secret', totp);

                setState(() {
                  key = decodedKey;
                  totpSecret = totp;
                });
                print(" Flutter AES Key (base64): ${base64.encode(key)}");
                print(" Flutter AES Key (HEX): ${key.map((b) => b.toRadixString(16).padLeft(2,
'0')).join()}");


                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(content: Text(' Конфігурацію зчитано успішно')),
                );
              } catch (e) {
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(content: Text(' Помилка обробки QR: $e')),
                );
              }
            }
          }

          @override
          Widget build(BuildContext context) {
            return Scaffold(
              resizeToAvoidBottomInset: true,
              appBar: AppBar(
                title: Text('WebSocket Chat'),
                actions: [
                  Padding(
                    padding: const EdgeInsets.only(right: 10.0),
                    child: Icon(
                      Icons.circle,
                      color: isConnected ? Colors.green : Colors.red,
                      size: 20,
```

```
      ),
    ),
  ],
),
body: SafeArea(
 child: Column(
    children: [
    Expanded(
      child: ListView.builder(
       itemCount: messages.length,
       itemBuilder: (context, index) {
        String message = messages[index];
        bool isYou = message.startsWith("You:");
        bool isSystem = message.startsWith("[Система]:");
        return ListTile(
         title: Text(
          message,
          style: TextStyle(
           color: isYou
              ? Colors.blue
              : (isSystem ? Colors.green : Colors.black),
          ),
         ),
        );
       },
      ),
    ),
    Padding(
     padding: const EdgeInsets.symmetric(horizontal: 8.0, vertical: 4.0),
     child: Row(
      children: [
       Expanded(
        child: TextField(
         controller: controller,
         decoration: InputDecoration(labelText: 'Enter message'),
        ),
       ),
       IconButton(
        icon: Icon(Icons.send),
        onPressed: sendMessage,
       ),
      ],
     ),
    ),
    Padding(
     padding: EdgeInsets.symmetric(horizontal: 8.0, vertical: 4.0),
     child: Wrap(
      spacing: 8,
```

```
            runSpacing: 4,
            alignment: WrapAlignment.center,
            children: [
              ElevatedButton(
                onPressed: isConnected ? disconnectWebSocket : connectWebSocket,
                child: Text(isConnected ? 'Disconnect' : 'Connect'),
                style: ElevatedButton.styleFrom(
                  backgroundColor: isConnected ? Colors.red : Colors.green,
                ),
              ),
              ElevatedButton(
                onPressed: scanQRConfiguration,
                child: Text('Сканувати конфігурацію'),
              ),
              ElevatedButton(
                onPressed: loadConfigFromClipboard,
                child: Text('З буфера'),
              ),
            ],
          ),
        ),
      ],
    ),
  ),
 );
 }
}

class QRScannerScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(title: const Text('Сканування QR')),
   body: MobileScanner(
    onDetect: (capture) {
     final barcode = capture.barcodes.first;
     final String? code = barcode.rawValue;
     if (code != null) {
      Navigator.pop(context, code);
     }
    },
   ),
  );
 }
}
```

Додаток Г. Лістинг коду адмін-додатку

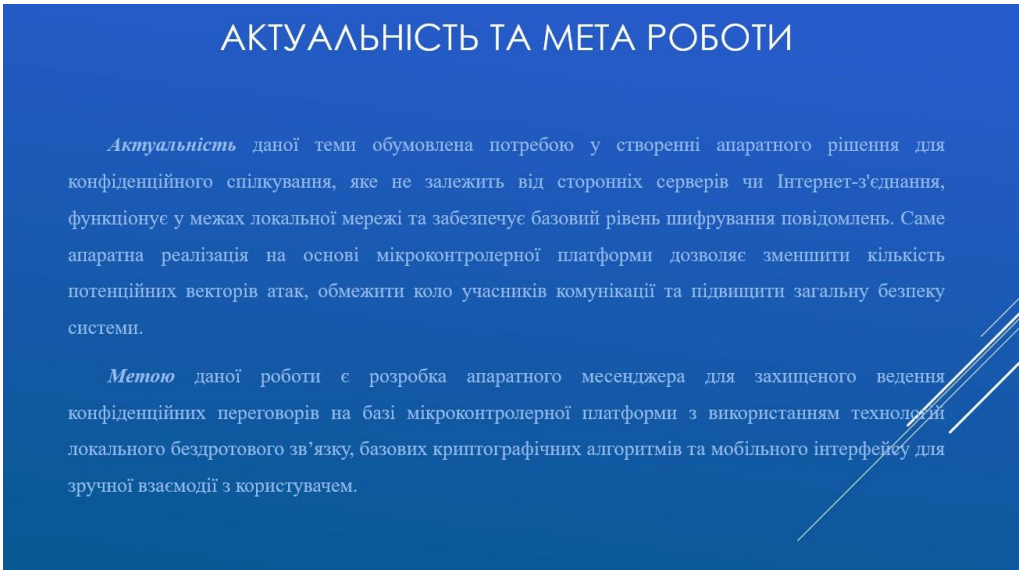
```
import 'package:flutter/material.dart';
import 'dart:convert';
import 'package:qr_flutter/qr_flutter.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:flutter/services.dart';
import 'package:web_socket_channel/web_socket_channel.dart';
import 'dart:math';

void main() {
  runApp(AdminApp());
}

class AdminApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Admin Panel',
      theme: ThemeData(primarySwatch: Colors.teal),
      home: AdminDashboard(),
    );
  }
}

class AdminDashboard extends StatefulWidget {
  @override
  _AdminDashboardState createState() => _AdminDashboardState();
}

class _AdminDashboardState extends State<AdminDashboard> {
  TextEditingController aesController = TextEditingController();
  TextEditingController totpController = TextEditingController();
  String? generatedQR;
  WebSocketChannel? channel;
  bool isConnected = false;
  final wsUrl = 'ws://192.168.4.1:81';

  @override
  void initState() {
    super.initState();
    _loadSavedConfig();
  }

  void _loadSavedConfig() async {
    final prefs = await SharedPreferences.getInstance();
    aesController.text = prefs.getString('aes_key') ?? '';
    totpController.text = prefs.getString('totp_secret') ?? '';
```

```
  _updateQR();
}

void _saveConfig() async {
  final prefs = await SharedPreferences.getInstance();
  await prefs.setString('aes_key', aesController.text);
  await prefs.setString('totp_secret', totpController.text);
  _updateQR();
}

void _updateQR() {
  final config = jsonEncode({
    "aes_key": aesController.text,
    "totp_secret": totpController.text,
  });
  setState(() {
    generatedQR = config;
  });
}

void _generateRandomKeys() {
  final randomBytes = List.generate(16, (index) => Random().nextInt(256));
  final randomKey = base64.encode(randomBytes);
  setState(() {
    aesController.text = randomKey;
    totpController.text = 'GEZDGNBVGY3TQOJQ';
    _saveConfig();
  });
}

void _copyConfigToClipboard() {
  final config = jsonEncode({
  "aes_key": aesController.text,
  "totp_secret": totpController.text,
  });
  Clipboard.setData(ClipboardData(text: config));
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text('Конфігурацію скопійовано в буфер обміну.')),
  );
}

void _connectAndSendToESP32() {
  try {
    channel = WebSocketChannel.connect(Uri.parse(wsUrl));
    setState(() => isConnected = true);
    final hexKey = _convertBase64ToHex(aesController.text);
    final payload = "SET_CONFIG:$hexKey;${totpController.text}";
    channel!.sink.add(payload);
```

```
            channel!.stream.listen((message) {
              ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('ESP32:
$message')));
            });
          } catch (e) {
            ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Помилка
підключення: $e')));
          }
        }

        void _sendRebootCommand() {
          try {
            if (channel == null) {
              channel = WebSocketChannel.connect(Uri.parse(wsUrl));
              setState(() => isConnected = true);
            }
            channel!.sink.add("REBOOT");
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(content: Text('Команда ребуту ESP32 відправлена')),
            );
          } catch (e) {
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(content: Text('Помилка при відправці REBOOT: $e')),
            );
          }
        }

        String _convertBase64ToHex(String base64Key) {
          final bytes = base64.decode(base64Key);
          return bytes.map((b) => b.toRadixString(16).padLeft(2, '0')).join();
        }

        @override
        void dispose() {
          channel?.sink.close();
          super.dispose();
        }

        @override
        Widget build(BuildContext context) {
          return Scaffold(
            appBar: AppBar(title: Text('Admin Dashboard')),
            body: Padding(
              padding: const EdgeInsets.all(16.0),
              child: SingleChildScrollView(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
```

```
            Text('AES Key (Base64):'),
            TextField(controller: aesController),
            SizedBox(height: 16),
            Text('TOTP Secret (Base32):'),
            TextField(controller: totpController),
            SizedBox(height: 16),
            ElevatedButton(
              onPressed: _saveConfig,
              child: Text(' Зберегти конфігурацію'),
            ),
            ElevatedButton(
              onPressed: _generateRandomKeys,
              child: Text(' Згенерувати нові ключі'),
            ),
            ElevatedButton(
              onPressed: _copyConfigToClipboard,
              child: Text(' Скопіювати в буфер обміну'),
            ),
            ElevatedButton(
              onPressed: _connectAndSendToESP32,
              child: Text(' Передати на ESP32'),
            ),
            ElevatedButton(
              onPressed: _sendRebootCommand,
              style: ElevatedButton.styleFrom(backgroundColor: Colors.red),
              child: Text(' Ребут ESP32'),
            ),
            SizedBox(height: 24),
            if (generatedQR != null)
              Center(
                child: Column(
                  children: [
                    Text('QR для передачі конфігурації:'),
                    SizedBox(height: 10),
                    QrImageView(
                      data: generatedQR!,
                      version: QrVersions.auto,
                      size: 200.0,
                    ),
                  ],
                ),
              )
          ],
        ),
      ),
    ),
  );
}
```

Додаток Д. Ілюстративний матеріал



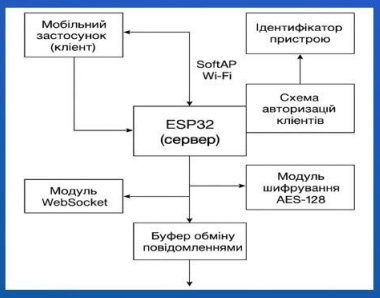
Месенджер	E2EE за замовчуванням	Анонімність	Відкритий код	Децентралізація	Потреба в номері телефону
Signal	Так	Ні	Так	Ні	Так
Telegram	Ні (тільки секретні)	Частково	Частково	Ні	Так
Threema	Так	Так	Частково	Ні	Ні
Session	Так	Так	Так	Так	Ні
Element	Так	Частково	Так	Так	Ні

АПАРАТНІ РІШЕННЯ В СФЕРІ ЗАХИЩЕНОЇ КОМУНІКАЦІЇ



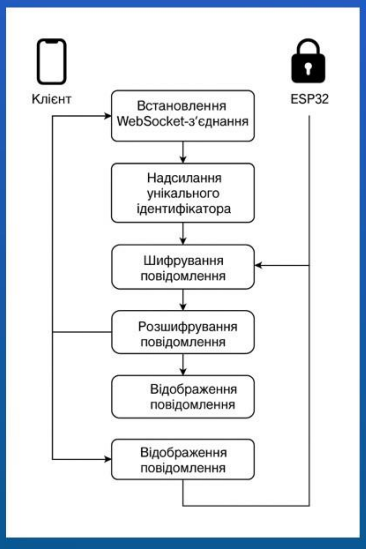

Пристрій	Шифрування	ОС / Платформа	Вартість (орієнтовна)	Призначення	Особливості
Blackphone 2	End-to-End (Silent Phone)	Silent OS (Android)	~$799	Бізнес/приватність	Фокус на приватність, відкритий код
Bittium Tough Mobile 2	AES, VPN, TPM	Android Secure Platform	~$1500–2000	Військове, урядове використання	Сертифікація, захист на рівні апаратури
Sectéra Edge	AES, Suite B	Windows CE	~$3000–5000+	Уряд США, спецслужби	Підтримка Top Secret даних
PGP-телефони (на базі Blackberry, Android)	PGP, OTR	Змінювана	~$1000+ під ключ	Бізнес, спецслужби	Часто кастомізовані, продаються через дистриб'юторів

АРХІТЕКТУРА СИСТЕМИ

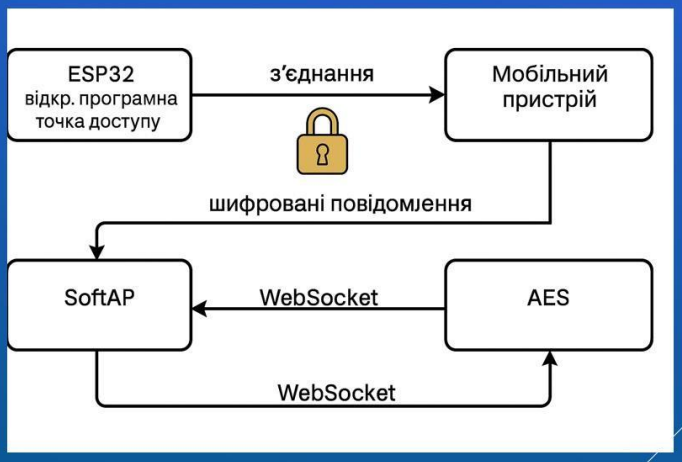


На зображеннях показано архітектура системи та взаємодія компонетів між собою.ESP32 створює локальну Wi-Fi точку доступу (SoftAP), до якої підключаються мобільні пристрої. З'єднання між ними відбувається через WebSocket, забезпечуючи постійний двосторонній зв'язок. Усі повідомлення передаються в зашифрованому вигляді за допомогою AES-128, що гарантує їх конфіденційність в межах цієї ізольованої мережі. Система працює автономно, без залежності від зовнішнього інтернету.

БЛОК-СХЕМА МОБІЛЬНОГО ЗАСТОСУНКУ (КЛІЄНТ)



ОБМІН ПОВІДОМЛЕННЯМИ



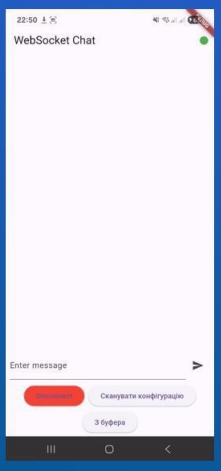
АЛГОРИТМ БЕЗПЕКИ

Алгоритм автентифікації на основі одноразових паролів (TOTP) працює у 7 кроків. Спочатку клієнт генерує OTP, використовуючи поточний час та спільний секретний ключ, що гарантує унікальність кожного пароля.

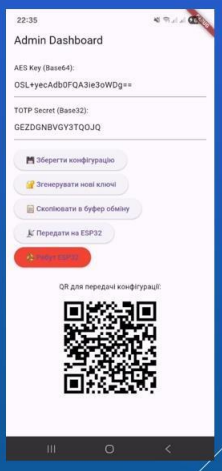
Одночасно ESP32 також генерує OTP, використовуючи той самий секретний ключ і поточний час для подальшої перевірки. Далі клієнт додає згенерований OTP до повідомлення та відправляє його на ESP32. Отримавши повідомлення, ESP32 виділяє з нього OTP і порівнює зі своїм власним, згенерованим на основі того ж секрету.

Якщо OTP співпадає, повідомлення приймається як валідне; якщо ні – відхиляється, і клієнту надсилається відповідь про помилку автентифікації. У випадку успішної автентифікації, ESP32 пересилає отримане повідомлення іншим перевіреним клієнтам, забезпечуючи захищений обмін інформацією в мережі.

ІНТЕРФЕЙСИ ДОДАТКІВ

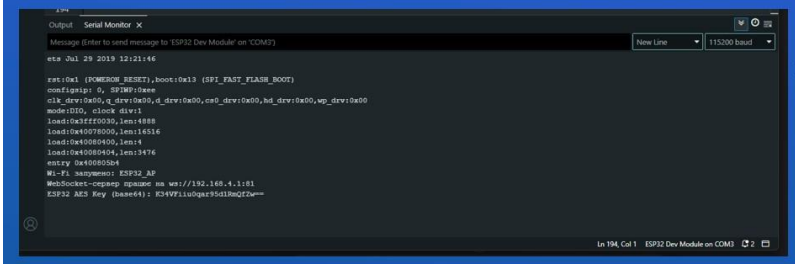


Клієнт

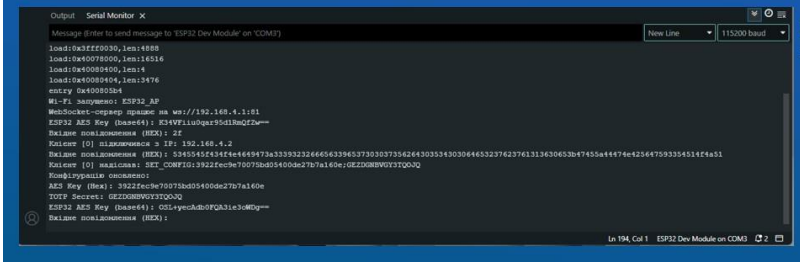


Адмін

ПЕРЕВІРКА ФУНКЦІОНУВАННЯ

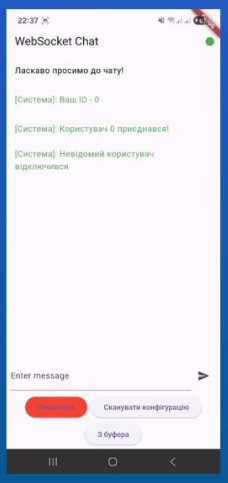


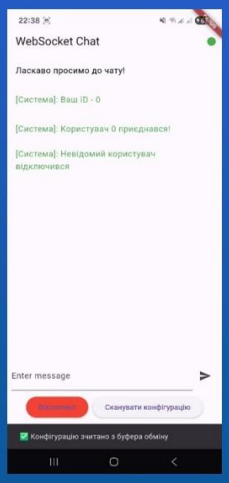
Ініціалізація мікроконтролерної платформи ESP32

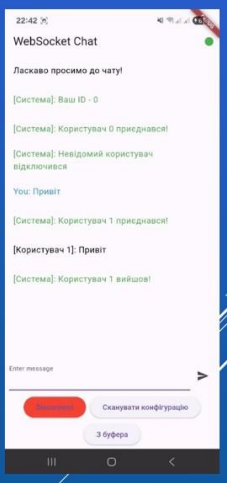


Відображення зміни конфігурації безпеки

ПЕРЕВІРКА ФУНКЦІОНУВАННЯ





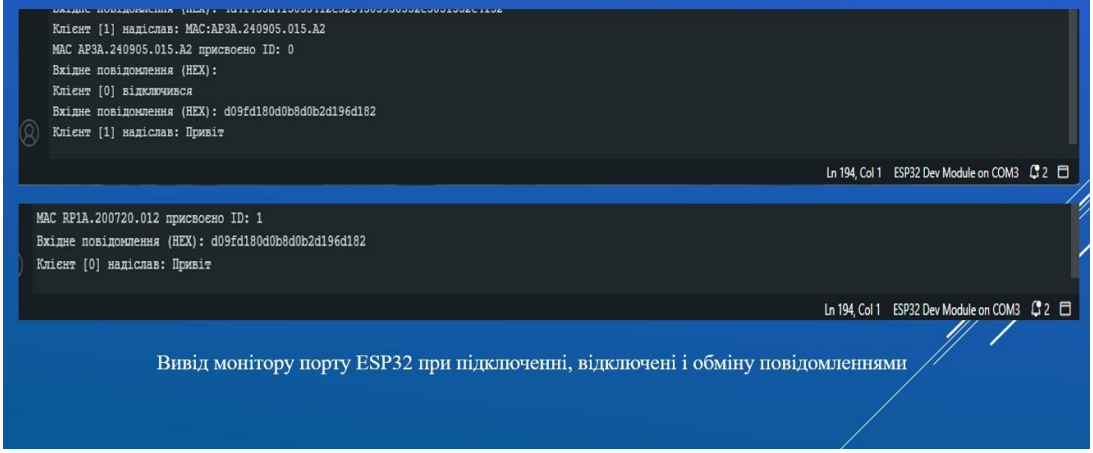


Підключення клієнта 1 до сервера

Результат оновлення конфігурації безпеки через буфер обміну

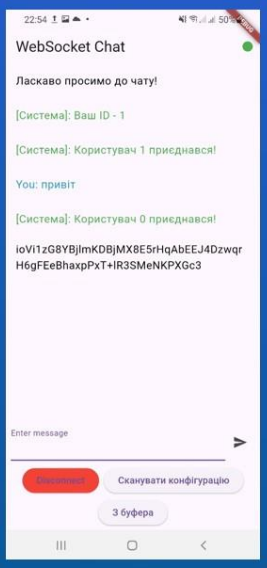
Відображення процесу відключення клієнта з поточного чату

ПЕРЕВІРКА ФУНКЦІОНУВАННЯ



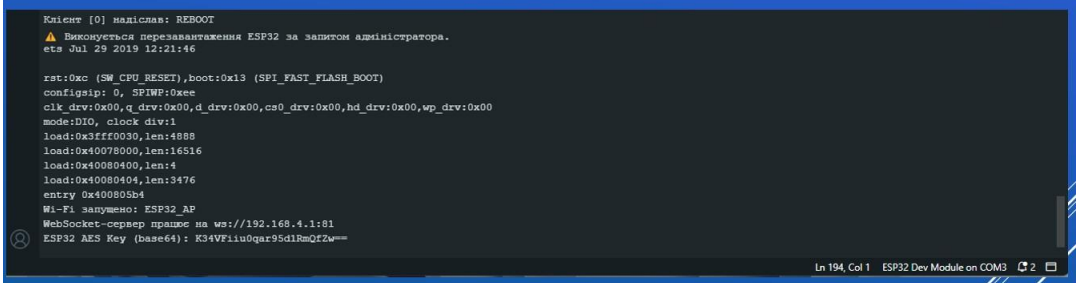
Вивід монітору порту ESP32 при підключенні, відключені і обміну повідомленнями

ПЕРЕВІРКА ФУНКЦІОНУВАННЯ



Інтерфейс клієнтського додатка 2 без проходження процедури оновлення конфігурації безпеки

ПЕРЕВІРКА ФУНКЦІОНУВАННЯ



Результат виконання віддаленого перезавантаження ESP32

ВИСНОВКИ

- Реалізовано локальну бездротову мережу SoftAP з можливістю підключення кількох клієнтів;
- Запущено WebSocket-сервер на ESP32, який обробляє повідомлення в реальному часі;
- Реалізовано централізоване шифрування повідомлень на ESP32 за алгоритмом AES-128;
- Інтегровано одноразові паролі TOTP для автентифікації користувачів перед шифруванням;
- Розроблено клієнтський застосунок на Flutter для введення повідомлень і взаємодії з системою;
- Створено адміністративний застосунок, який дозволяє оновлювати ключі, секрети та конфігурацію в реальному часі;
- Проведено демонстраційне тестування, яке підтвердило стабільність і коректну роботу системи в умовах практичного використання.

Таким чином, реалізований прототип демонструє, що навіть на базі доступної мікроконтролерної платформи можна створити ефективний, захищений і автономний засіб для ведення конфіденційного обміну повідомленнями.

Додаток Е. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка апаратного месенджера для захисту ведення конфіденційних переговорів на базі мікроконтролерної платформи

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
факультет менеджменту та інформаційної безпеки
Гр. 1КІТС-21б

Керівник доцент Шиян А.А. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	98,74%
Загальна схожість	1,26%

Аналіз звіту подібності (відмітити потрібне)

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

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

Автор _____ Кравчук В.О.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Допустити до захисту**

_____
_____
_____
_____

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

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

Додаток Ж. Протокол функціонального тестування

№ Тест-кейсу	Назва тесту	Очікуваний результат	Фактичний результат	Статус
1	Ініціалізація сервера ESP32	Успішний запуск, створення точки доступу Wi-Fi та WebSocket-сервера.	Відповідає очікуваному.	Успішно
2	Оновлення конфігурації з адмін-додатку	Сервер приймає та застосовує новий AES-ключ та секрет TOTP.	Відповідає очікуваному	Успішно
3	Обмін повідомленнями між авторизованими клієнтами	Повідомлення успішно шифруються, передаються та дешифруються.	Відповідає очікуваному.	Успішно
4	Спроба перегляду повідомлення неавторизованим клієнтом	Повідомлення відображається у вигляді нечитабельного шифротексту.	Відповідає очікуваному.	Успішно
5	Віддалене перезавантаження сервера	ESP32 виконує перезавантаження після отримання команди.	Відповідає очікуваному.	Успішно