МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут автоматики і електротехніки
Кафедра комп'ютерних технологій та інформаційної безпеки

«ДОПУЩЕНИЙ ДО ЗАХИСТУ»
Завідувач кафедри комп'ютерних технологій та інформаційної безпеки, к.т.н., доцент
_____ С.М. Нужний
«17» 06 2026р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти «бакалавр»

ДОСЛІДЖЕННЯ МЕХАНІЗМІВ НАСКРІЗНОГО ШИФРУВАННЯ В P2P-ЧАТІ

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 125 «Кібербезпека та захист інформації»
Освітньо-професійна програма: «Системи технічного захисту інформації, автоматизація її обробки»

Виконав: студент 4381 групи
Петрушин Олексій Олексійович _____
(прізвище та ініціали) (підпис)

Керівник роботи: к.т.н., доцент кафедри комп'ютерних технологій та інформаційної безпеки
Сірівчук А.С. _____
(прізвище та ініціали) (підпис)

Миколаїв – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут автоматики та електротехніки

Кафедра комп'ютерних технологій та інформаційної безпеки

Спеціальність **125«Кібербезпека та захист інформації»**
Освітня програма **«Системи технічного захисту інформації, автоматизація її обробки»**

ЗАТВЕРДЖУЮ
Гарант освітньої програми, к.т.н.,
доцент, доцент кафедри КТІБ
_____ С.М. Нужний
«10» 08 2026р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти «бакалавр»

Студенту Петрушину Олексію Олексійовичу

1. Тема роботи: Дослідження механізмів наскрізного шифрування в P2P-чаті

Керівник роботи к.т.н., доцент кафедри комп'ютерних технологій та інформаційної безпеки, Сірівчук Андрій Сергійович

Затверджені наказом ректора №286/г2 від 22.04.2026

2. Термін подання роботи: 5.06.2026
3. Вихідні дані по роботі: Децентралізована система/ наявність наскрізного шифрування/ робота в мереж P2P

4. Перелік питань, що належать до розробки (найменування розділів): Призначення та область застосування; Аналіз сучасних методів захисту повідомлень у месенджерах; Проєктування системи P2P-чату з наскрізним шифруванням; Експериментальне дослідження та оцінка ефективності розробленої системи
5. Перелік презентаційних матеріалів: Титульний лист; Мета та завдання роботи; Теоретичні основи захищеного P2P-обміну повідомленнями; Аналіз існуючих рішень; Архітектура системи; Програмна реалізація; Результати тестування безпеки; Аналіз стійкості до пасивного аналізу за допомогою ПЗ Wireshark; Активний аналіз (MITM); Висновок

6. Консультант - внутрішній рецензент роботи

Прізвище, ініціали та посада	Дата подання роботи на рецензію	Дата отримання рецензії	Підпис
Перезрій В.І. професор, д.т.н.	2.06.26	5.06.26	

7. Дата видачі завдання 10.02.2026

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання	Примітка
1	Вибір теми, визначення мети, завдань, об'єкта та предмета дослідження	10.02.2026	
2	Попередній варіант оглядових розділів, підбір і опрацювання списку використаних джерел	20.03.2026	
3	Попередній варіант повної КР	15.04.2026	
4	Складання ЄДКІ зі спеціальності на ступінь «бакалавра»	16.04.2026 14.05.2026	
5	Попередній варіант презентації	29.05.2026	
6	Подання на кафедру КТІБ тексту остаточного варіанту роботи, підписаного її керівником в електронному форматі разом із заявами щодо самостійності виконання роботи та ідентичності друкованої та електронної версій роботи (Додатки 1 і 2 «Порядку здійснення заходів з перевірки робіт на наявність текстових збігів/ідентичності/схожості із використанням програмно-технічних засобів», який введений в дію наказом ректора НУК за №20 від 20.01.2020) Отримання внутрішньої рецензії	05.06. 2026	не пізніше, ніж за 14 діб до захисту
7	Отримання висновку керівника; Отримання висновку кафедри щодо наявності / відсутності плагіату в роботі Попередній захист роботи на засіданні кафедри КТІБ Висновок кафедри про КР, допуск до захисту	10.06.2026	
8	Отримання зовнішньої рецензії	15.06.2026	
9	Підготовка презентації та доповіді	17.06.2026	
10	Подання на кафедру КТІБ: – друкованого і зшитого варіанту пояснювальної записки до кваліфікаційної роботи та/або (за рішенням кафедри) електронного варіанту в форматі *.pdf – друкованого варіанту презентації (в 5-ти екземплярах в разі захисту оф-лайн, в одному екземплярі в разі захисту он-лайн (за рішенням кафедри)) та електронного варіанту презентації; – зовнішньої рецензії; – документів щодо відсутності плагіату і передачі авторських прав; – файлу-опису кваліфікаційної роботи (згідно Додатку до наказу ректора НУК за №287-уч від 19.05.2020 р.).	17.06.2026	
11	Захист кваліфікаційної роботи	24 - 25. 06.2026	

Студент _____ (підпис) О.О. Петрушин (ПІБ)

Керівник роботи _____ (підпис) А.С. Сірівчук (ПІБ)

АНОТАЦІЯ

Петрушин О.О. Дослідження механізмів наскрізного шифрування в *P2P*-чаті. Кваліфікаційна робота на здобуття ступеня вищої освіти «бакалавр» за спеціальністю 125 «Кібербезпека та захист інформації» галузі знань 12 «Інформаційні технології». – Національний університет кораблебудування імені адмірала Макарова, Миколаїв, 2026.

Пояснювальна записка: 46 с., 8 рис., 1 таблицю, 23 посилання.

У кваліфікаційній роботі розглянуто питання дослідження та розробки децентралізованої системи обміну повідомленнями з наскрізним шифруванням на основі однорангової (P2P) архітектури. Проведено аналіз сучасних підходів до захищених комунікацій, принципів інформаційної безпеки та криптографічних механізмів, що використовуються для забезпечення конфіденційності, цілісності й автентичності даних.

У роботі реалізовано програмний прототип *P2P*-чату мовою програмування Go із використанням гібридної криптографічної схеми, що поєднує алгоритми *X25519 (ECDH), Ed25519, HKDF-SHA256* та *AES*-256-*GCM*. Розроблена архітектура забезпечує відсутність єдиної точки відмови, захист від атак типу «людина посередині», підтримку наскрізного шифрування та ефективну взаємодію між вузлами мережі. Користувацький інтерфейс реалізовано у вигляді кросплатформенного текстового консольного клієнта.

Проведене тестування із застосуванням інструментів Gosec та Wireshark підтвердило високу стійкість системи до пасивного й активного аналізу мережевого трафіку. Результати дослідження засвідчили ефективність запропонованих архітектурних і криптографічних рішень та можливість використання розробленої системи як основи для побудови захищених засобів корпоративного і приватного обміну інформацією.

Ключові слова: наскрізне шифрування, *E2EE*, децентралізація, *P2P*-чат, *Go*, *ECDH*, *X25519*, *AES-GCM*.

ANNOTATION

Petrushyn O.O. Research of end-to-end encryption mechanisms in P2P chat. Qualification work for the degree of higher education "Bachelor" in the specialty 125 "Cybersecurity and information protection" field of knowledge 12 "Information technologies". - Admiral Makarov National University of Shipbuilding, Mykolaiv, 2026.

Explanatory note: 46 p., 8 figures, 1 table, 23 references.

The qualification work considers the issue of research and development of decentralized messaging systems with end-to-end encryption based on peer-to-peer (P2P) architecture. An analysis of modern approaches to secure communications, principles of information security and cryptographic mechanisms used to ensure confidentiality, integrity and authenticity of data is carried out.

The work implements a software prototype of P2P chat based on Go programming using a hybrid cryptographic scheme that corresponds to the X25519 (ECDH), Ed25519, HKDF-SHA256 and AES-256-GCM algorithms. The developed architecture provides the absence of a single point of failure, protection against man-in-the-middle attacks, support for end-to-end encryption and effective interaction between network nodes. The user interface is implemented in the form of a cross-platform text console client.

Testing using the Gosec and Wireshark tools confirmed the high resistance of the system to passive and active analysis of network traffic. The results of the study demonstrated the effectiveness of the proposed architectural and cryptographic solutions and the possibility of using the developed system as a basis for building secure means of corporate and private information exchange.

Keywords: end-to-end encryption, E2EE, decentralization, P2P chat, Go, ECDH, X25519, AES-GCM.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AEAD – Authenticated Encryption with Associated Data)

CBC – Cipher Block Chaining)

DHT – Distributed Hash Table

E2EE – End-to-End Encryption

ECB – Electronic Codebook

ECDH – Elliptic Curve Diffie-Hellman

GUI – raphical User Interface

HKDF – HMAC-based Key Derivation Function

ICE – Interactive Connectivity Establishment

KDF – Key Derivation Function

MAC – Message Authentication Code

NAT – Network Address Translation

P2P – Peer-to-Peer

STUN – Session Traversal Utilities for NAT

TUI – Text User Interface

TURN – Traversal Using Relays around NAT

X3DH – Extended Triple Diffie-Hellman

ВСТУП

У сучасну епоху цифровізації та стрімкого розвитку технологій інтернет-комунікацій безпека персональних даних та конфіденційність листування стали одними з найважливіших пріоритетів інформаційного суспільства. Зростання кількості кібератак, витоків даних із централізованих серверів, а також спроби несанкціонованого доступу до приватних розмов з боку третіх сторін зумовлюють високий попит на захищені системи обміну повідомленнями.

Традиційні клієнт-серверні архітектури, попри свою популярність, мають суттєвий недолік – наявність єдиної точки відмови та компрометації (*Single Point of Failure*) в особі центрального сервера. Альтернативою цьому є архітектура рівноправних вузлів (*Peer-to-Peer*, *P2P*), яка виключає посередників з процесу передачі даних. Проте, відсутність довіреного централізованого вузла створює нові виклики для забезпечення безпеки, автентифікації учасників та керування ключами.

Найбільш ефективним методом захисту інформації в таких умовах є наскрізне шифрування (*End-to-End Encryption*, *E2EE*), де ключі шифрування відомі лише кінцевим користувачам. Дослідження, аналіз та практична реалізація механізмів *E2EE* у *P2P*-середовищах є актуальним науково-практичним завданням, спрямованим на створення децентралізованих, стійких до злому та незалежних засобів комунікації.

Метою дипломної роботи є теоретичне обґрунтування, аналіз та практична реалізація механізмів наскрізного шифрування в архітектурі *P2P*-чату для забезпечення високого рівня конфіденційності, автентичності та цілісності даних у децентралізованому середовищі.

Для досягнення поставленої мети передбачено вирішення низки взаємопов'язаних завдань, що включають:

– аналіз сучасного стану та проблем безпеки в існуючих децентралізованих системах обміну повідомленнями;

– комплексне дослідження математичних основ і криптографічних алгоритмів наскрізного шифрування;

– розробити модель загроз та сформулювати вимоги до безпеки *P*2*P*-чату;

– спроектувати стійку архітектуру системи обміну повідомленнями без використання центрального сервера

– реалізувати програмний прототип розробленого чату.

Кваліфікаційна робота відповідає наступним фаховим компетентностям:

ФК-2 – Здатність використовувати інформаційні технології, сучасні методи і моделі кібербезпеки та системи захисту інформації;

ФК-7 – Здатність забезпечувати захист інформації в інформаційних та інформаційно-комунікаційних системах згідно встановленої політики кібербезпеки й захисту інформації;

ФК-8 – Здатність застосовувати методи та засоби криптографічного захисту інформації на об’єктах інформаційної діяльності.

РОЗДІЛ 1. ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

1.1 Призначення системи *P*2*P*-чату з наскрізним шифруванням

Головним призначенням розроблюваної інформаційної системи є забезпечення безпечного, конфіденційного та незалежного від сторонніх сервісів обміну текстовими повідомленнями та медіафайлами між двома або більше користувачами (вузлами мережі) у режимі реального часу [1].

На відміну від традиційних месенджерів, функціонування цієї системи базується на поєднанні децентралізованої архітектури *Peer-to-Peer* (*P*2*P*) та криптографічного протоколу наскрізного шифрування (*End-to-End Encryption* – *E*2*EE*).

З функціональної точки зору система призначена для вирішення таких ключових завдань:

– пряма взаємодія користувачів (*Peer-to-Peer*): Встановлення безпосереднього зв'язку між клієнтськими пристроями для передачі даних без проходження через централізовані сервери обробки повідомлень [2]. Це нівелює ризик блокування, цензурування або витоку даних з єдиної бази даних;

– забезпечення абсолютної конфіденційності (*E*2*EE*): Зашифрування інформації безпосередньо на пристрої відправника та її розшифрування виключно на пристрої отримувача. Дані передаються через транзитні мережеві вузли у вигляді криптографічного шуму, що унеможливлює їх перехоплення атакою типу «людина посередині» (*Man-in-the-Middle*, *MitM*) [3, 4];

– керування криптографічними ключами на стороні клієнта: Генерація, збереження та автентифікація ключів шифрування відбуваються локально на пристроях користувачів, що виключає передачу закритих ключів у мережу.

– гарантування цілісності та автентичності повідомлень: Захист інформації від модифікації або підміни під час транспортування за допомогою механізмів імітозахисту та автентифікованого шифрування.

Цільове призначення системи полягає у наданні користувачам інструменту комунікації, який гарантує суверенність їхніх персональних даних, на відміну від комерційних засобів комунікації, де оператор сервісу має технічну можливість (або може бути змушений під тиском) збирати метадані, аналізувати графік спілкування або передавати інформацію третім сторонам. У розроблюваному *P*2*P*-чаті закладено неможливість будь-якого зовнішнього контролю за змістом бесіди чи фактом її ведення з боку третіх осіб [2].

1.2 Область застосування та користувачі системи

Децентралізована архітектура у поєднанні з наскрізним шифруванням визначає специфіку застосування розроблюваної системи. Оскільки *P*2*P*-чат не залежить від центральних серверів та інфраструктури конкретних IT-корпорацій, сфера його впровадження охоплює сегменти, де вимоги до конфіденційності, автономності та відмовостійкості є критично високими [1].

Основні області застосування системи:

– корпоративний сектор та комерційна таємниця: Організація внутрішніх каналів зв'язку для обміну конфіденційною інформацією, фінансовими звітами, стратегічними планами чи технологічними секретами. Використання *P*2*P*-чату виключає ризик промислового шпигунства через компрометацію хмарних серверів сторонніх провайдерів [2];

– робота в умовах обмеженого або порушеного зв'язку (Критична інфраструктура): Комунікація всередині локальних або закритих відомчих мереж (наприклад, підприємства енергетики, транспортні вузли, екстрені служби). Оскільки для роботи *P*2*P*-чату достатньо прямої мережевої видимості між пристроями (через локальну мережу *Wi-Fi*, *Ethernet* або *Mesh*-мережі) [2], система залишається працездатною навіть у разі повного відключення глобального інтернету чи магістральних каналів зв'язку.

– державний сектор та правозахисна діяльність: Забезпечення каналів зв'язку для обміну даними, що підлягають особливому правовому захисту

(наприклад, адвокатська таємниця, журналістські розслідування, комунікація правозахисних організацій);

– особиста безпечна комунікація: Використання широким колом користувачів, які прагнуть захистити своє приватне життя від масового збору метаданих, таргетованої реклами та цифрового профайлінгу.

Цільові групи користувачів (Аудиторія системи). Узагальнений портрет користувача системи – це суб'єкт інформаційних відносин, для якого суверенність даних та відсутність посередників є пріоритетом. Користувачів системи можна класифікувати на такі категорії:

– кінцеві споживачі: Фізичні особи, які володіють базовими навичками роботи з цифровими пристроями, але потребують інструменту для приватного спілкування без ризику цензури чи витоку даних;

– корпоративні користувачі: Співробітники компаній, менеджери вищої ланки та аналітики, які використовують систему на робочих місцях для обговорення комерційних питань без залучення стороннього хмарного програмного забезпечення;

– системні адміністратори та IT-спеціалісти: Користувачі, які можуть розгортати та інтегрувати розроблений *P2P*-протокол у вже існуючу локальну інфраструктуру підприємства, адаптуючи його під специфічні корпоративні потреби.

Завдяки відсутності потреби у реєстрації через централізовані ідентифікатори (такі як номери телефонів чи облікові записи *Google*/*Apple*), коло користувачів системи розширюється за рахунок осіб, які вимагають повного або часткового рівня анонімності на етапі створення з'єднання.

1.3 Огляд існуючих рішень

Для визначення оптимального технологічного стеку та архітектури розроблюваної системи необхідно провести аналіз існуючих аналогів, які позиціонують себе як захищені засоби комунікації. Сучасні рішення можна

розділити на дві основні категорії: централізовані месенджери з підтримкою *E2EE* та повністю децентралізовані (*P2P*) системи.

Серед централізованих та федеративних системи з наскрізним шифруванням можна виділити *Signal* та *Matrix*.

Signal Вважається золотим стандартом у сфері криптографічного захисту повідомлень завдяки використанню однойменного *Signal Protocol* (поєднання *X3DH* та *Double Ratchet*) [5]. Месенджер забезпечує найвищий рівень конфіденційності вмісту чатів. Проте, *Signal* є повністю централізованою системою. Він залежить від серверів компанії, а для реєстрації користувача обов'язково потрібен номер телефону, що негативно впливає на анонімність та створює єдину точку відмови у разі блокування серверів.

Matrix це децентралізована, але не *P2P*-система. Вона базується на федеративній архітектурі, де користувачі реєструються на різних серверах, які потім синхронізуються між собою [6]. Для шифрування використовується протокол *Olm*/*Megolm* (адаптація *Double Ratchet*) [7]. Головний мінус – система все одно потребує постійно діючих серверів, що ускладнює її розгортання в ізольованих локальних мережах без серверної інфраструктури.

До децентралізованих (*Peer-to-Peer*) систем можна віднести *Tox* (клієнти *qTox*, *uTox*) та *Briar*.

Tox це повністю децентралізований *P2P*-месенджер, який працює без жодного центрального сервера. Для пошуку вузлів та побудови мережі використовується розподілена хеш-таблиця (*DHT*) на основі алгоритму *Kademlia* [2, 8]. Усі дані шифруються за допомогою криптографічної бібліотеки *NaCl* (алгоритми *Curve*25519 [9], *XSalsa*20, *Poly*1305). *Tox* є найближчим аналогом розроблюваної системи. Проте його головний недолік – високе навантаження на мережу та акумулятор мобільних пристроїв через постійну підтримку зв'язків у *DHT*-мережі, а також складність реалізації групових чатів без сервера.

Briar є надійним *P2P*-месенджером, який був створений для журналістів та активістів. Він може передавати повідомлення через мережу *Tor* (якщо є

інтернет), або напряму через *Wi-Fi* та *Bluetooth* (якщо інтернету немає). Забезпечує високий рівень анонімності та стійкості до цензури. Головне обмеження *Briar* – він заточений під мобільні пристрої *Android* та специфічні сценарії використання (на кшталт мітингів чи зон стихійного лиха), а обмін повідомленнями можливий лише тоді, коли обидва контакти одночасно перебувають у мережі або передають дані через спільних знайомих (*Mesh*-принцип) [10].

Для системного аналізу існуючих рішень та визначення вимог до власної системи сформулюємо порівняльну таблицю (табл. 1.1) за ключовими критеріями безпеки та архітектури.

Таблиця 1.1 Порівняння існуючих рішень та розроблюваної системи

Критерій порівняння	*Signal*	*Matrix* (*Element*)	*Tox*	*Briar*	Розроблювана система (*P2P*-чат)
Тип архітектури	Централізована	Федеративна	*P2P*	*P2P*	*P2P*
Наявність серверів	Так (обов'язково)	Так (сервери федерації)	Ні	Ні	Ні
Наскрізне шифрування (*E2EE*)	Так	Так	Так	Так	Так
Прив'язка до ідентифікатора (тел. / *email*)	Так (номер телефону)	Так (обліковий запис)	Ні (унік. *ID*)	Ні (унік. *ID*)	Ні (унік. крипто. *ID*)

Продовження таблиці 1.1

Робота в ізольованій локальній мережі (*LAN*)	Ні	Ні	Так (локальний пошук)	Так (*Wi-Fi* / *Bluetooth*)	Так (пряме *TCP* з'єднання)
Динамічне оновлення ключів (*Ratchet*)	Так	Так	Ні (статична сесія)	Так	Ні

Аналіз показує, що існуючі централізовані рішення (*Signal*) мають відмінну криптографію, але вразливі на рівні інфраструктури. Децентралізовані аналоги (*Tox*, *Briar*) вирішують проблему серверів, але є складними у розгортанні, перевантажують мережеві інтерфейси або мають обмеження щодо платформ.

Таким чином, розробка полегшеного *P2P*-чату, який фокусується на прямому з'єднанні вузлів в межах доступних мереж (*LAN*/*WAN*) із базовим наскрізним шифруванням, є виправданим кроком для створення швидкого, автономного та незалежного інструменту комунікації.

Висновки до розділу

За результатами дослідження встановлено, що головне цільове призначення системи полягає у забезпеченні абсолютно конфіденційного, безпечного та незалежного від сторонніх сервісів обміну текстовими повідомленнями та медіафайлами в режимі реального часу. Поєднання децентралізованої архітектури *Peer-to-Peer* та протоколу наскрізного шифрування дозволяє повністю нівелювати ризики блокування, цензурування, витоку персональних даних або реалізації атак типу «людина посередині».

РОЗДІЛ 2. АНАЛІЗ СУЧАСНИХ МЕТОДІВ ЗАХИСТУ ПОВІДОМЛЕНЬ У МЕСЕНДЖЕРАХ

2.1 Поняття та принципи наскрізного шифрування

Наскрізне шифрування (*End-to-End Encryption*, *E2EE*) – це метод захисту передачі даних, при якому інформація шифрується на пристрої відправника (першому кінцевому вузлі) і розшифровується виключно на пристрої отримувача (другому кінцевому вузлі). Ключовою особливістю *E2EE* є те, що жоден посередник – будь то інтернет-провайдер, магістральний оператор зв'язку, зловмисник у локальній мережі чи навіть центральний сервер розробника – не має технічної можливості отримати доступ до відкритого тексту повідомлень [11].

Щоб зрозуміти фундаментальну важливість *E2EE*, необхідно порівняти його зі звичайним транспортним шифруванням (*Encryption-in-Transit*), яке використовується у більшості стандартних клієнт-серверних додатків (наприклад, через протоколи *HTTPS*/*TLS*).

При транспортному шифруванні дані захищені лише на шляху від клієнта до сервера. На самому сервері повідомлення розшифровується, обробляється (або зберігається в базі даних) і знову зашифровується для відправки іншому клієнту. Це створює серйозні ризики:

– компрометація сервера або бази даних призводить до витоку листування всіх користувачів;

– адміністратори сервера або треті особи (за рішенням суду чи внаслідок інсайдерської атаки) мають прямий доступ до вмісту чатів.

Натомість наскрізне шифрування перетворює проміжну інфраструктуру на «сліпий» канал передачі даних (так званий *Dumb Pipe*). Сервер або проміжні *P2P*-вузли бачать лише зашифровані бінарні дані (шифротекст) та метадані (наприклад, *IP*-адреси, час відправки), які необхідні для маршрутизації.

Функціонування надійної системи наскрізного шифрування базується на чотирьох фундаментальних принципах інформаційної безпеки [1]:

– конфіденційність (*Confidentiality*): Доступ до змісту повідомлення мають лише законні учасники діалогу. Реалізується за допомогою стійких алгоритмів симетричного шифрування [3, 12], ключі до яких згенеровані безпосередньо на кінцевих пристроях користувачів;

– цілісність даних (*Data Integrity*): Гарантія того, що повідомлення не було змінене, підмінене або частково видалене під час транспортування мережею. Для цього використовується автентифіковане шифрування з приєднанням *MAC*;

– автентифікація (*Authentication*): Впевненість у тому, що співрозмовник є саме тією особою, за яку себе видає. У системах *E*2*EE* це досягається за допомогою криптографії з відкритим ключем (асиметричної криптографії), де кожен користувач має пару ключів: відкритий (*public*) для ідентифікації та шифрування, і закритий (*private*) для розшифрування та створення цифрового підпису [12];

– неможливість відмови від авторства (*Non-repudiation*): Відправник повідомлення не може заперечити факт його відправки, оскільки кожне повідомлення підписується його унікальним закритим криптографічним ключем, який зберігається в захищеному локальному сховищі пристрою.

У контексті децентралізованих *P*2*P*-чатів принципи *E*2*EE* зазнають певних модифікацій. Оскільки в системі відсутній довірений центр сертифікації ключів, автентифікація користувачів покладається на прямий обмін відбитками ключів (*fingerprints* / *QR*-коди) або використання механізмів павутини довіри (*Web of Trust*). Це робить математичну стійкість алгоритмів та коректність генерації ключів головним рубежем оборони системи.

2.2 Архітектура та особливості *P*2*P*-мереж

Архітектура *Peer-to-Peer* (*P*2*P*), або рівноправна (однорангова) мережа – це концепція побудови розподілених інформаційних систем, у якій всі учасники (вузли, що називаються пірами або бенчами) мають рівні права, обов'язки та виконують одночасно функції як клієнта, так і сервера. На відміну від традиційної клієнт-серверної моделі, де централізований сервер є єдиним постачальником ресурсів і координатором взаємодії, у *P*2*P*-мережі обмін даними (текстовими повідомленнями, файлами тощо) здійснюється безпосередньо між кінцевими пристроями користувачів [2].

Для глибокого розуміння особливостей *P*2*P*-мереж необхідно розглянути їх відмінності від класичних систем організації мережевої взаємодії.

Клієнт-серверна модель (*Client-Server*): Характеризується чітким поділом ролей. Клієнти ініціюють запити, а виділений сервер їх обробляє. Така схема є легкою в адмініструванні та масштабуванні на початкових етапах, але має критичний недолік – наявність єдиної точки відмови (*Single Point of Failure*). Якщо сервер виходить з ладу, піддається кібератаці (наприклад, *DDoS*) або блокується регулятором, уся система повністю припиняє функціонування. Крім того, власник сервера володіє абсолютним контролем над даними користувачів.

Федеративна модель (*Federated*): Є проміжним варіантом (наприклад, екосистема *Email* або *Matrix*), де замість одного сервера існує безліч незалежних серверів, які можуть взаємодіяти між собою. Користувач залежить від конкретного сервера, на якому зареєстрований.

Децентралізована *P*2*P*-модель: Повністю виключає поняття постійно діючого сервера з ланцюжка комунікації. Мережа формується динамічно самими користувачами. Кожен новий вузол, підключаючись до мережі, не лише споживає її ресурси (отримує повідомлення), а й надає свої (маршрутизує трафік, підтримує таблиці зв'язків).

За способом організації зв'язків між вузлами та механізмами пошуку інформації *P*2*P*-мережі поділяють на три основні типи [2]:

– неструктуровані *P*2*P*-мережі: Вузли зв'язуються випадковим чином. Пошук інших учасників відбувається методом "повені" (*flooding*) – запит надсилається всім сусіднім вузлам, ті пересилають своїм сусідам і так далі. Метод є простим, але створює колосальне надлишкове навантаження на канали зв'язку і не гарантує знаходження адресата;

– структуровані *P*2*P*-мережі (на базі *DHT*): Мережа будується за чіткими топологічними правилами. Для пошуку вузлів використовується Розподілена хеш-таблиця (*Distributed Hash Table*, *DHT*), найчастіше на основі алгоритму *Kademlia* (використовується в *BitTorrent*, *Tox*). Кожному вузлу та файлу присвоюється унікальний хэш-ідентифікатор, а пошук адресата відбувається за логарифмічний час $O(\log N)$, що робить мережу високоефективною;

– гібридні *P*2*P*-мережі: Поєднують децентралізовану передачу даних із використанням допоміжних серверів. Наприклад, сервери можуть використовуватися виключно на етапі авторизації або пошуку *IP*-адреси співрозмовника, після чого зв'язок переходить у чистий *P*2*P* (модель, близька до ранніх версій *Skype*).

Реалізація *P*2*P*-чату з наскрізним шифруванням відкриває унікальні переваги, але водночас створює низку серйозних технічних викликів [2]:

– проблема обходу *NAT* (*NAT Traversal*): Більшість сучасних користувачів перебувають у приватних локальних мережах за роутерами та брандмауерами, отримуючи динамічні або "сірі" *IP*-адреси (технологія *NAT*). Два пристрої за різними *NAT* не можуть встановити пряме *TCP*/*UDP* з'єднання без сторонньої допомоги. Для вирішення цієї проблеми у *P*2*P* використовуються спеціальні протоколи: *STUN* (*Session Traversal Utilities for NAT*) для визначення публічної *IP*-адреси та порту, *ICE* (*Interactive Connectivity Establishment*) для пошуку оптимального шляху з'єднання, а у крайніх випадках – *TURN* (*Traversal Using Relays around NAT*), коли дані ретранслюються через допоміжний проміжний вузол [13].

– динамічність мережі (*Churn Rate*): У *P*2*P*-мережі вузли постійно підключаються, відключаються, змінюють *IP*-адреси або переходять в офлайн.

Система повинна гнучко реагувати на втрату зв'язку, підтримувати актуальні списки маршрутизації та коректно завершувати або відновлювати криптографічні сесії.

– асинхронний обмін повідомленнями: У клієнт-серверній моделі, якщо отримувач офлайн, сервер зберігає повідомлення в базі даних і доставляє його пізніше. У чистому *P*2*P*, якщо отримувач не в мережі, надіслати повідомлення напряму неможливо. Це вимагає розробки специфічних архітектурних компромісів (наприклад, збереження повідомлення в черзі на стороні відправника до моменту появи отримувача, або використання *Mesh*-принципів, де зашифроване повідомлення тимчасово зберігається на транзитних вузлах-посередниках без ризику розшифрування).

Таким чином, проектування *P*2*P*-чату вимагає не лише інтеграції криптографічних інструментів *E*2*EE*, а й побудови надійного мережевого рівня, здатного ефективно вирішувати завдання адресації, ідентифікації та обходу мережевих обмежень в умовах повної відсутності довіреного центрального сервера.

2.3 Огляд криптографічних алгоритмів, що використовуються у чатах

Для практичної реалізації наскрізного шифрування (*E*2*EE*) в архітектурі *P*2*P*-чату криптографічні алгоритми об'єднуються у гібридну систему. Асиметрична криптографія використовується для безпечного узгодження спільного секрету між вузлами, які не довіряють каналу зв'язку, а симетрична – для безпосереднього та швидкого зашифрування текстових повідомлень і файлів.

Головною проблемою симетричного шифрування є безпечна передача секретного ключа від відправника до отримувача. У *P*2*P*-мережі, де немає центрального сервера, цю проблему вирішують алгоритми асиметричного розподілу ключів.

Протокол Діффі-Гелмана (*Diffie-Hellman*, *DH*): Перша та класична схема, що дозволяє двом сторонам створити спільний секретний ключ через незахищений канал зв'язку. Безпека базується на складності обчислення дискретного логарифма у скінченному полі [3, 12].

Протокол Діффі-Гелмана на еліптичних кривих (*Elliptic Curve Diffie-Hellman*, *ECDH*): Сучасна модифікація класичного протоколу. Замість мультиплікативних груп великих цілих чисел використовуються групи точок на еліптичній кривій. Головна перевага *ECDH* порівняно з класичним *DH* або *RSA* – вища криптостійкість при значно меншому розмірі ключа. Наприклад, ключ *ECDH* довжиною 256 біт (крива *Curve*25519 або *secp*256*k*1) забезпечує такий самий рівень захисту, як 3072-бітний ключ *RSA*. Це критично для *P*2*P*-систем, оскільки зменшує обсяг мережевого трафіку та навантаження на процесор під час ініціалізації з'єднання [9, 12, 14].

Симетричні шифри використовують один і той самий ключ для зашифрування та розшифрування. Вони працюють у тисячі разів швидше за асиметричні, тому ідеально підходять для потокової передачі повідомлень у чаті.

Advanced Encryption Standard (*AES*): Найпоширеніший у світі блочний шифр із довжиною ключа 128, 192 або 256 біт. Для сучасних месенджерів критично важливим є режим роботи шифру. Використання застарілих режимів (наприклад, *ECB* або *CBC*) є небезпечним без додаткових засобів контролю цілісності. У сучасних архітектурах застосовують *AES*-*GCM* (*Galois*/*Counter Mode*) – *AEAD*. Одночасно із шифруванням тексту *AES*-*GCM* генерує імітовставку (автентифікаційний тег) [15, 16]. Якщо зловмисник спробує змінити хоча б один біт у шифротексті під час передачі в *P*2*P*-мережі, отримувач миттєво виявить це на етапі дешифрування і відхилить повідомлення.

*ChaCha*20-*Poly*1305: Інший популярний *AEAD*-алгоритм, де потоковий шифр *ChaCha*20 комбінується з генератором автентифікаційних тегів *Poly*1305. Він часто використовується як альтернатива *AES* (наприклад, у протоколі *TLS*

1.3 чи *VPN WireGuard*). На пристроях, які не мають апаратного прискорення для *AES* (деякі старі смартфони чи *IoT*-модулі), *ChaCha*20 працює значно швидше [17].

Хеш-функції перетворюють вхідні дані довільного обсягу у фіксовану строку байтів. Вони є невід'ємною частиною архітектури безпеки чату.

Сімейство *SHA*-2 (зокрема *SHA*-256): Використовується для генерації унікальних ідентифікаторів повідомлень, створення цифрових підписів, а також для отримання "відбитків ключів" (*Key Fingerprints*). У *P*2*P*-чатах відбиток (хеш) відкритого ключа користувача часто виступає в ролі його унікального мережевого *ID* (адреси), за якою його знаходять інші піри [3].

HKDF (*HMAC-based Extract-and-Expand Key Derivation Function*): Спеціалізована функція розгортання ключів на основі хешування. Вона бере базовий секрет, отриманий після процедури *ECDH*, і "розгортає" його у кілька незалежних ключів (наприклад, один ключ для шифрування повідомлень за допомогою *AES*, інший – для підтвердження автентичності) [18].

2.4 Аналіз протоколів безпечного обміну повідомленнями

Для побудови цілісної системи наскрізного шифрування (*E*2*EE*) недостатньо просто обрати окремі алгоритми шифрування чи узгодження ключів; їх необхідно об'єднати у високорівневий криптографічний протокол. Такий протокол регламентує чітку послідовність дій: як саме ініціалізується сесія, як автентифікуються учасники, яким чином оновлюються ключі та як обробляються повідомлення, що прийшли із запізненням.

Проведемо аналіз найбільш поширених та технологічно досконалих протоколів захищеної комунікації, які використовуються у сучасних інформаційних системах.

2.4.1 *Signal Protocol* (колишній *Axolotl*). Протокол *Signal* на сьогодні вважається індустріальним стандартом для *E*2*EE*. Його архітектура базується на

поєднанні двох основних криптографічних примітивів: протоколу початкового узгодження ключів *X3DH* (*Extended Triple Diffie-Hellman*) та алгоритму оновлення ключів *Double Ratchet* [5].

X3DH (*Extended Triple Diffie-Hellman*): Цей протокол вирішує проблему асинхронної ініціалізації сесії. Він дозволяє встановити спільний секретний ключ, навіть якщо один із користувачів перебуває в офлайні. Для цього використовуються заздалегідь згенеровані та завантажені на сервер одноразові відкриті ключі (*PreKeys*). Протокол виконує серію з трьох або чотирьох операцій Діффі-Гелмана між різними типами ключів (ідентифікаційними, підписаними та одноразовими), що гарантує високий рівень автентифікації та захисту від підробки.

Double Ratchet Algorithm: Після того, як *X3DH* встановив початковий секрет, *Double Ratchet* бере на себе керування сесійними ключами. Він поєднує симетричний храповик (на основі *KDF*-ланцюжків) для кожного надісланого повідомлення та асиметричний храповик (на основі обміну новими ключами *DH*) для кожної зміни напрямку діалогу. Це забезпечує дві найважливіші властивості безпеки: пряму секретність (*Forward Secrecy*) та захист від компрометації у майбутньому (*Post-Compromise Security*).

Головний недолік у контексті *P2P*: Протокол *Signal* критично залежить від наявності централізованого сервера для збереження *PreKeys*. Без серверної інфраструктури логіка *X3DH* руйнується, що робить його складним для прямої інтеграції в чисті *P2P*-системи без серйозних модифікацій.

2.4.2 Протоколи *Olm* та *Megolm* (екосистема *Matrix*). Протокол *Olm* – це криптографічна реалізація алгоритму *Double Ratchet*, адаптована для децентралізованої федеративної мережі *Matrix* [6, 7]. За своєю суттю він дуже схожий на *Signal Protocol* і призначений для шифрування чатів "тет-а-тет".

Для групових чатів у *Matrix* використовується окремий протокол – *Megolm*. Оскільки класичний *Double Ratchet* у великих групах генерує колосальне навантаження на мережу (потрібно підтримувати окрему сесію з кожним учасником групи), *Megolm* використовує інший підхід. Кожен учасник

групи створює власний ланцюжок ключів (*Outbound Session*) і ділиться його початковим станом з іншими учасниками через захищені канали *Olm*. Кожне повідомлення шифрується симетричним ключем, який прокручується по симетричному храповику.

Головний недолік у контексті *P2P*: Хоча *Megolm* ефективно вирішує проблему групових чатів, він жертвує властивістю *Post-Compromise Security*. Якщо зловмисник компрометує стан ланцюжка ключів одного з учасників, він зможе розшифровувати всі його майбутні повідомлення у цій групі, доки користувач вручну не оновить сесію. Крім того, протокол все ще орієнтований на федеративні сервери, а не на пряму *P2P*-взаємодію сокетів.

2.4.3 Криптографічний стек бібліотеки *NaCl* / *libsodium.* У багатьох чистих *P2P*-системах (наприклад, у месенджері *Tox* або мережевому стеку *libp2p* [20]) замість складних асинхронних протоколів типу *Signal* використовується низькорівневий, але надзвичайно швидкий і надійний криптографічний стек *NaCl* (*Networking and Cryptography library*) [19] або його сучасне відгалуження *libsodium*.

Замість постійного динамічного оновлення ключів для кожного повідомлення, тут застосовується концепція статичної захищеної сесії. При встановленні з'єднання два вузли виконують одноразовий обмін ключами за протоколом *ECDH* (*Curve*25519) та узгоджують сесійний ключ. Усі подальші повідомлення шифруються за допомогою алгоритму автентифікованого шифрування (наприклад, *ChaCha*20-*Poly*1305 або *AES-GCM*), де унікальність кожного пакету забезпечується суворою інкрементною зміною лічильника *nonce*.

Висновки до розділу

На основі порівняння наскрізного шифрування (*E2EE*) та транспортного шифрування встановлено, що лише *E2EE* забезпечує повну конфіденційність

даних на всьому шляху передачі, усуваючи ризики компрометації на проміжних вузлах. Дослідження принципів інформаційної безпеки в однорангових мережах показало важливість математичної стійкості криптографічних алгоритмів і локального контролю ключів через відсутність централізованого довіреного центру. Аналіз *P2P*-архітектури підтвердив її високу відмовостійкість і відсутність єдиної точки відмови, водночас виявивши потребу в застосуванні протоколів обходу *NAT* (*STUN, ICE, TURN*) та механізмів підтримки асинхронної взаємодії. Також обґрунтовано ефективність гібридної криптографічної моделі на основі *ECDH* для узгодження ключів і *AEAD*-шифрів (*AES-GCM, ChaCha20-Poly1305*) для захищеного обміну даними, а аналіз сучасних рішень показав, що протоколи на кшталт *Signal* значною мірою залежать від серверної інфраструктури для підтримки офлайн-користувачів.

РОЗДІЛ 3. ПРОЄКТУВАННЯ СИСТЕМИ P2P-ЧАТУ З НАСКРІЗНИМ ШИФРУВАННЯМ

3.1 Архітектура системи

Програмна архітектура децентралізованого застосунку базується на поєднанні поділу системи на ізольовані шари (*Layered Architecture*) та подієво-орієнтованої моделі взаємодії (*Event-Driven Architecture*) через асинхронні канали зв'язку. Оскільки кожен запущений екземпляр системи (вузол) є одночасно і клієнтом, і сервером, архітектура виключає концепцію централізованого координатора, покладаючи логіку управління сесіями та даними безпосередньо на логічні блоки самого вузла (рис. 3.1).

Рисунок 3.1 – Схематичне зображення архітектури застосунку

Програма розділена на три рівні абстракції, що взаємодіють між собою через строго визначені інтерфейси та структури даних [21]:

– рівень інтерфейсу користувача (*TUI Layer*): Реалізований у пакеті *internal*/*tui*. Він відповідає за термінальний графічний інтерфейс застосунку (*Text User Interface*), рендеринг списку чатів, відображення історії повідомлень та зчитування вводу оператора. Цей шар повністю відокремлений від мережевої логіки.

– рівень управління вузлом та сесіями (*Core* / *Peer Layer*): Реалізований у пакеті *internal*/*peer*. Це центральний архітектурний компонент, який ініціалізує хост *libp2p* [20], управляє життєвим циклом мережевих потоків (*SafeStream*),

координує процеси криптографічного рукостискання та здійснює маршрутизацію повідомлень;

– рівень абстракції даних та збереження станів (*Data Domain Layer*): Реалізований у пакеті *internal/defs*. Він описує доменні моделі даних: структуру повідомлення (*Message*), структуру кімнати чату (*ChatData*) та менеджер сховища (*ChatStorage*).

Взаємодія між рівнем інтерфейсу (фронтендом) та рівнем вузла (бекендом) здійснюється асинхронно за допомогою архітектурного шаблону «Брокер» (*Broker*). Структура *defs.Broker* містить два канали:

– *UpdateOnFront* – канал, через який мережевий рівень повідомляє інтерфейс про зміну статусу з'єднання, надходження нових повідомлень або необхідність оновити екран;

– *UpdateOnBack* – канал, через який інтерфейс надсилає бекенду сигнали про необхідність ініціалізувати нове підключення або відправити повідомлення, що перебувають у статусі очікування (*Pending*).

Основним об'єктом, що описує стан вузла, є структура *peer.Peer* (рис. 3.2).

```
type Peer struct {
    Host     host.Host
    MyPeerID string

    PrivKey        crypto.PrivKey
    SessionPrivKey *ecdh.PrivateKey
    SessionPubKeys []byte

    Streams    map[string]*SafeStream
    StreamsMut sync.RWMutex

    Chats  *defs.ChatStorage
    Broker defs.Broker
}
```

Рисунок 3.2 - Вигляд структури вузла

Її внутрішня архітектура містить такі критично важливі компоненти:

a) компонент криптографічної ідентифікації (*PrivKey* та *loadPrivateKey*): Під час старту системи застосунок звертається до файлу *key*. Якщо файл відсутній, компонент генерує довгострокову пару ключів за алгоритмом *Ed*25519, маршалізує їх у байтовий потік та зберігає на диск із правами доступу 0600 (доступ має тільки власник). Цей ключ передається в опцію *libp*2*p.Identity*, формуючи незмінний *Peer ID* вузла;

b) мережевий хост (*host.Host*): Ініціалізує прослуховування *IPv*4-інтерфейсів за протоколом *TCP* на порту, вказаному в змінній оточення (*PORT*). Він також реєструє єдиний обробник протоколу системи *SetStreamHandler*(*PROTOCOL*, ...) для фільтрації та обробки вхідних *P*2*P*-з'єднань;

c) пул захищених потоків (*Streams*): Геш-таблиця (*map*), захищена за допомогою м'ютексу взаємного виключення читанння/запису *sync.RWMutex*. Вона асоціює *Peer ID* віддалених користувачів із об'єктами *SafeStream*. Об'єкт *SafeStream* є архітектурною обгорткою над мережевим потоком *network.Stream*, яка додатково інкапсулює унікальний 32-байтний симетричний ключ *AES* даної сесії та канал синхронізації ініціалізації *isInit*;

d) диспетчер фонових рутин (*Background Workers*): Архітектура реалізує паралельне виконання завдань за допомогою двох фонових горутин, що запускаються під час ініціалізації хоста:

– *readBroker* – безперервно прослуховує канал *UpdateOnBack*. При отриманні об'єкта чату вона ініціює з'єднання, отримує мережевий потік і виконує ітеративне вичитування черги повідомлень у зворотному порядку (*slices.Backward*), відправляючи в потік усі повідомлення, що мають статус *Pending*.

– *refreshConnections* – періодичний таймер, який здійснює моніторинг сховища чатів. Якщо статус якогось чату переходить у стан *Failed* (з'єднання втрачено), рутина автоматично переводить його в статус *Connecting* та відправляє брокеру запит на відновлення зв'язку.

Взаємодія між двома вузлами в межах виділеного потоку суворо регламентована та розділена на два етапи.

Етап рукостискання (*Handshake Stage*): При відкритті потоку (вхідного чи вихідного) функція *streamsHandler* негайно записує в сокет серіалізовану структуру *sessionPubKeys*. Ця структура містить ефемерний відкритий ключ *X*25519 та його цифровий підпис, створений довгостроковим ключем *Ed*25519. Вхідна горутина *readFromStream* зчитує цей перший пакет, витягує довгостроковий відкритий ключ із *Peer ID* віддаленого користувача, верифікує підпис, обчислює спільний секрет та через *HKDF-SHA*256 формує 32-байтний ключ симетричного шифрування, записуючи його в *SafeStream*. Канал *isInit* розблоковується.

Етап обміну повідомленнями (*Streaming Stage*): Оскільки *TCP* є потоковим протоколом, який не зберігає меж повідомлень, в архітектуру чату інтегровано рівень префіксації. Перед відправкою шифротексту до нього додається префікс довжини (*utils.AddLengthPrefixToMessage*). При зчитуванні, функція *utils.ReadMessageWithLengthPrefix* спочатку дізнається точний розмір пакету з префіксу, вичитує саме цю кількість байтів із сокету і передає масив байт на розшифрування. Це гарантує цілісність десеріалізації повідомлень.

Така архітектура забезпечує повну потокобезпечність завдяки активному використанню м'ютексів (*sync.RWMutex*), виключає гонки даних, а також автоматично відновлює з'єднання без втрати повідомлень завдяки буферизованим чергам статусів *Pending*.

3.2 Модель взаємодії вузлів у *P*2*P*-мережі

Взаємодія рівноправних вузлів у системі побудована на базі децентралізованого протоколу, що виключає потребу в центральному сервері. Життєвий цикл з'єднання координується паралельними процесами всередині кожного вузла й послідовно проходить кілька ключових етапів.

Під час запуску застосунку вузол зчитує довгостроковий приватний ключ з локального файлу ідентифікації або автоматично генерує новий, формуючи свій унікальний мережевий паспорт – *Peer ID*. Далі ініціалізується мережевий хост, який відкриває *TCP*-порт для прослуховування інтерфейсів і реєструє єдиний обробник протоколу системи, переходячи в режим очікування вхідних підключень.

Встановлення логічного потоку (*Stream*) між двома конкретними учасниками відбувається асинхронно. Активний режим ініціюється відправником, коли він пише повідомлення: об'єкт чату потрапляє в канал брокера *UpdateOnBack*, звідки фонова рутина витягує адресу цільового піра і відкриває новий вихідний потік даних. Пасивний режим відпрацьовує на стороні отримувача, коли його хост фіксує вхідний запит за сигнатурою протоколу й передає його внутрішньому обробнику. Одразу після відкриття потоку вузли виконують криптографічне рукостискання (*Handshake*).

Під час безпосереднього обміну повідомленнями текст шифрується симетричним ключем із додаванням унікального випадкового вектора ініціалізації (*nonce*) для кожного пакету. Для запобігання злипанню пакетів у мережевому *TCP*-сокеті застосунок обчислює точну довжину зашифрованого масиву і додає її як префікс перед відправкою. Приймаюча рутина спочатку зчитує префікс довжини, суворо вибирає потрібну кількість байтів із сокету й передає їх дешифратору. Після перевірки цілісності повідомлення зберігається в пам'яті й через канал *UpdateOnFront* миттєво відправляється на екран користувача.

Для стійкості до раптових зникнень пірів із мережі застосовано фоновий монітор з'єднань, що працює циклічно з таймером у 250 мілісекунд. Якщо через обрив зв'язку виникає мережева помилка, потік закривається, а статус чату переводиться в стан помилки. Фоновий воркер автоматично виявляє такий чат, змінює його статус на режим підключення та відправляє брокеру запит на відновлення зв'язку. Це змушує систему автоматично повторити весь цикл взаємодії з генерацією нових ефемерних ключів без ручного втручання.

3.3 Механізм генерації та обміну ключами

Для забезпечення конфіденційності передачі даних реалізовано гібридну криптографічну схему, яка виконує асиметричне рукостискання для створення спільного секрету з його подальшим розгортанням у симетричний сесійний ключ. Процес обміну ключами інтегровано безпосередньо в момент відкриття з'єднання між вузлами та інкапсульовано у функції *getKeyToStream*. Процедура криптографічного узгодження параметрів безпеки ініціюється автоматично для кожного нового або відновленого з'єднання.

На першому етапі процедури застосунок викликає функцію генерації сесійних ключів, яка ініціює роботу з еліптичною кривою *X*25519. За допомогою криптографічно стійкого генератора псевдовипадкових чисел створюється тимчасовий закритий сесійний ключ та відповідний йому відкритий ефемерний ключ. Для підтвердження автентичності згенерованого ефемерного секрету та захисту від атак типу «людина посередині» вузол бере масив байтів свого відкритого ключа *X*25519 та підписує його за допомогою довгострокового приватного ключа *Ed*25519, який завантажується з файлу ідентифікації хоста. Надіслані дані пакуються у структуру сесійних ключів, серіалізуються у формат *JSON*, що містить поля для відкритого ключа та його цифрового підпису, і записуються у відкритий мережевий потік відправника.

Приймаюча сторона зчитує цей пакет із сокету і негайно виконує перевірку підпису. Оскільки відкритий довгостроковий ключ *Ed*25519 співрозмовника математично зв'язаний з його мережевим ідентифікатором *Peer ID*, одержувач використовує його для дешифрування підпису та порівняння з надісланим ефемерному ключем *X*25519. Якщо цифровий підпис не проходить верифікацію, потік негайно закривається з реєстрацією помилки безпеки, що повністю нівелює спроби підміни вузла або перехоплення транзиту даних.

Після успішної автентифікації обидва вузли незалежно один від одного викликають метод обчислення спільного секрету. На основі власного приватного сесійного ключа та отриманого відкритого ключа віддаленого вузла

обчислюється спільний секрет Діффі-Гелмана. Отриманий масив даних передається у функцію розгортання ключів, де за допомогою стандартизованого алгоритму *HKDF* на базі хеш-функції *SHA*-256 система формує фінальний 32-байтний симетричний ключ для алгоритму *AES*.

На завершальному етапі отриманий симетричний ключ зберігається в структурі, що відповідає з’єднанню з віддаленим вузлом, після чого внутрішній сигнальний канал ініціалізації закривається. Це слугує повідомленням для інших фонових процесів програми про те, що етап криптографічного рукостискання успішно завершено, і система переходить у режим готовності до безпечного автентифікованого шифрування текстових повідомлень користувача.

3.4 Алгоритм шифрування та дешифрування повідомлень

Безпосередній захист конфіденційної інформації під час її транзиту через децентралізовані канали зв'язку покладається на симетричний криптографічний модуль. Процес обробки текстових даних побудований на використанні симетричного блочного шифру *Advanced Encryption Standard* із довжиною ключа 256 біт, який функціонує в режимі автентифікованого шифрування з приєднаними даними *Galois*/*Counter Mode*. Вибір даного режиму зумовлений його здатністю забезпечувати одночасно як конфіденційність повідомлення, так і строгий контроль його цілісності та автентичності без залучення додаткових засобів обчислення кодів автентифікації.

Алгоритм шифрування повідомлення ініціюється на боці відправника у функції *EncryptMessage* після того, як користувач вводить текст у термінальному інтерфейсі застосунку. На вхід функції передається сформований раніше 32-байтний симетричний сесійний ключ та масив байтів відкритого тексту. Спочатку система створює новий екземпляр блочного шифру *AES* на основі переданого ключа, після чого ініціалізує режим *GCM*. Для гарантування унікальності криптографічного результату для кожного окремого

повідомлення застосунок звертається до системного генератора псевдовипадкових чисел *rand.Reader* для створення випадкового дванадцятибайтного вектора ініціалізації, або нонсенсу (*nonce*). Повторне використання одного й того самого нісенсу з однаковим симетричним ключем є критичною вразливістю, тому випадкова генерація для кожного пакету є обов'язковою архітектурною вимогою системи.

Сам процес шифрування виконується методом *gcm.Seal*. Цей метод приймає відкритий текст, обчислює для нього 16-байтний автентифікаційний тег (імітовставку) і виконує операцію порозрядного виключного АБО. Особливістю реалізації системи є те, що згенерований *nonce* автоматично записується на початок результуючого слайсу байтів, виступаючи префіксом для самого шифротексту та приєднаного тегу. Отриманий суцільний масив байтів повертається з криптографічного модулю, після чого до нього додається префікс загальної довжини пакету для коректного транспортування через *TCP*-потік.

Алгоритм дешифрування та верифікації повідомлення виконується на стороні отримувача в ізольованій фоновій рутині за допомогою функції *DecryptMessage*. Після того, як застосунок за допомогою префікса довжини вичитує точну кількість байтів зашифрованого пакету із сокету, цей масив передається на вхід дешифратора разом із локальним симетричним сесійним ключем. Функція повторно ініціалізує блочний шифр *AES* та режим *GCM*. На першому етапі обробки вхідного масиву програма перевіряє довжину отриманих даних: якщо вона є меншою за стандартний розмір нісенсу, процес негайно переривається з поверненням помилки про занадто короткий шифротекст.

За умови валідної довжини пакету застосунок виконує операцію слайсингу, розділяючи отриманий масив на дві частини. Перші 12 байтів відсікаються і зчитуються як вектор ініціалізації, а решта масиву ідентифікується як безпосередній шифротекст із приєднаним автентифікаційним тегом. Далі викликається метод *gcm.Open*, який виконує

зворотні математичні перетворення. Цей метод одночасно дешифрує текст і здійснює сувору перевірку цілісності за допомогою інтегрованого тегу. Якщо під час транзиту в *P*2*P*-мережі зловмисник або завади в каналі зв'язку змінили бодай один біт інформації, обчислений тег не співпаде з надісланим, метод поверне критичну помилку, а скомпрометоване повідомлення буде повністю знищено системою без виведення на екран користувача. У разі успішного проходження верифікації функція повертає чистий масив байтів відкритого тексту, який конвертується у рядок і відображається в активному вікні чату.

3.5 Реалізація клієнтського інтерфейсу чату

Важливою складовою проєкту розроблюваного децентралізованого месенджера є проектування та розробка інтерфейсу користувача. Традиційні графічні інтерфейси (*GUI*) часто створюють надлишкове навантаження на систему та вимагають значної кількості сторонніх залежностей. З огляду на це для клієнтської частини програми було обрано архітектуру текстового інтерфейсу користувача (*Text User Interface*, *TUI*). Цей підхід дозволяє запускати застосунок безпосередньо всередині будь-якого терміналу, забезпечуючи високу швидкодію, низьке споживання оперативної пам'яті та повну кросплатформенність. Інтерфейс користувача наведено на рисунках 3.3 та 3.4.

Візуальна структура інтерфейсу побудована на базі модульних контейнерів та гнучкої сітки, що адаптується до розмірів вікна терміналу в реальному часі. Керування застосунком та навігація між елементами інтерфейсу реалізовані за допомогою гарячих клавіш терміналу, що дозволяє користувачу швидко перемикатися між кімнатами розмов без використання комп'ютерної миші.

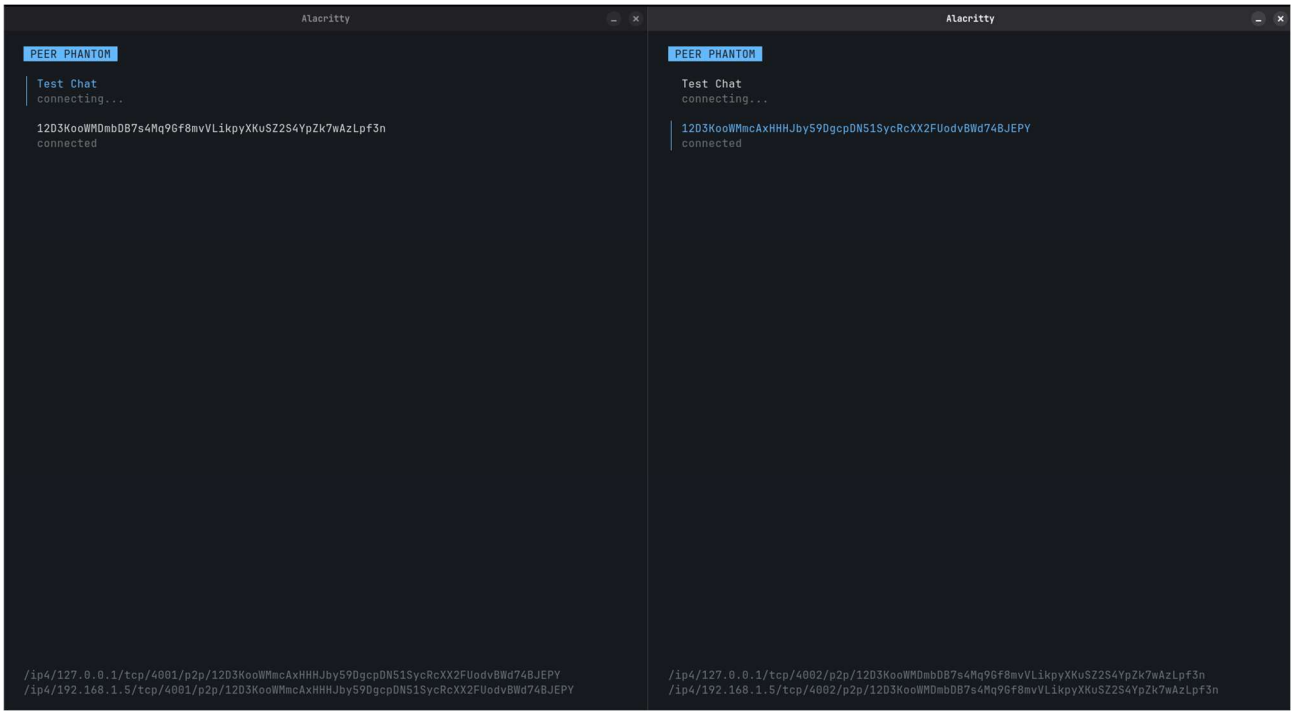

Рисунок 3.3 – Головне меню застосунку з відображенням списків чатів та адрес для підключення

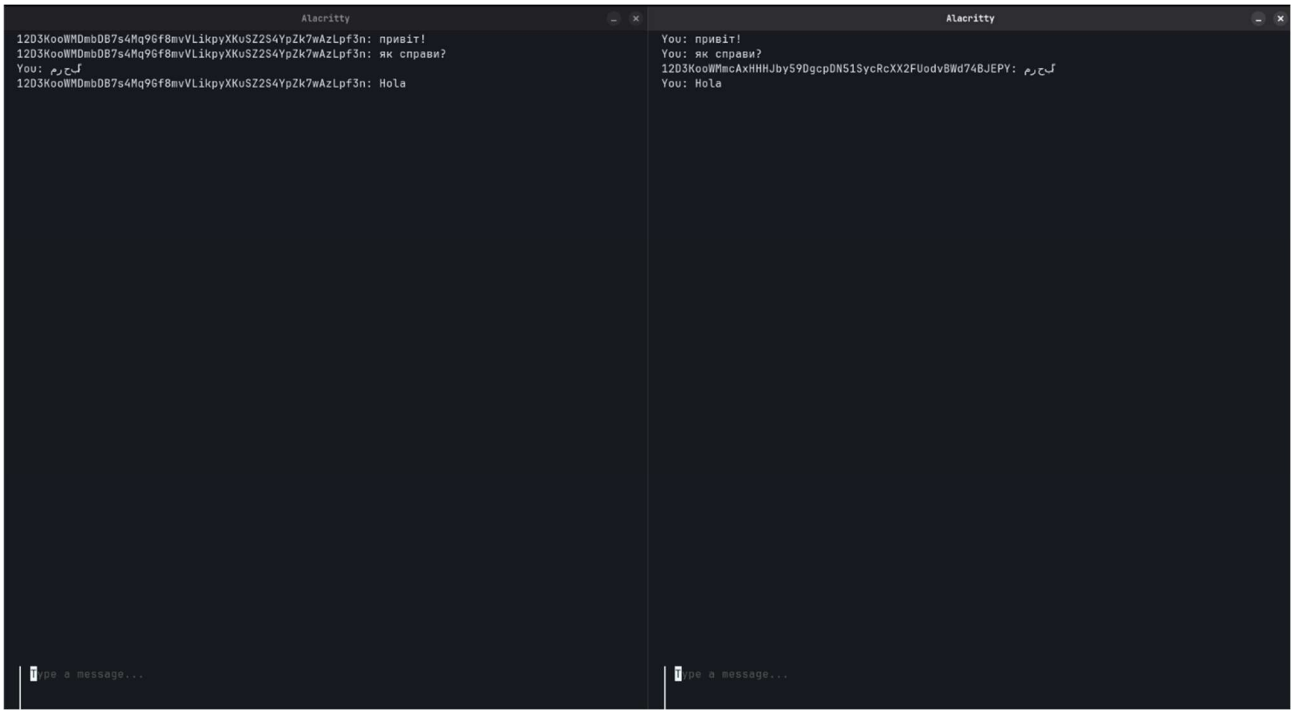

Рисунок 3.4 – Інтерфейс чату

Архітектурною особливістю реалізації клієнтського інтерфейсу є його повна ізоляція від мережевого ядра та криптографічного модуля за допомогою

подієво-орієнтованого шаблону проектування. Інтерфейс функціонує в окремому головному потоці виконання і не має прямого доступу до низькорівневих сокетів або процедур шифрування. Вся взаємодія здійснюється через структуру асинхронного брокера каналів. Коли користувач вводить повідомлення та натискає клавішу підтвердження, інтерфейсний рівень створює локальний об'єкт повідомлення, додає його до слайсу відображення зі статусом очікування доставки та миттєво відправляє вказівник на чат у вихідний канал брокера для подальшої обробки мережевим рушієм.

Оновлення графічного стану терміналу також відбувається асинхронно під впливом вхідних сигналів. Спеціальний фоновий процес інтерфейсу безперервно прослуховує чергу подій від брокера. При отриманні повідомлення про зміну статусу чату, встановлення успішного шифрованого з'єднання або при надходженні нового розшифрованого вхідного пакету, інтерфейс ініціює операцію повного перемальовування відповідного текстового віджета. Такий підхід повністю усуває проблему блокування інтерфейсу під час тривалих мережевих таймаутів або виконання ресурсомістких криптографічних операцій рукостискання, гарантуючи плавність роботи та високу швидкодію консольного додатку.

Висновки до розділу

У даному розділі спроєктовано та реалізовано архітектуру, мережеві протоколи, криптографічні механізми та користувацький інтерфейс децентралізованого *P2P*-чату. Поєднання ізольованої та подієво-орієнтованої архітектур забезпечило розмежування рівнів системи й неблокуючу взаємодію між інтерфейсом та мережевою логікою через асинхронний брокер подій. Для підтримки стабільної роботи в одноранговому середовищі реалізовано автоматичне керування з’єднаннями, чергами повідомлень і відновленням зв’язку.

РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ

4.1. Методика тестування безпеки

Для підтвердження надійності та стійкості розробленого програмного забезпечення було сформовано комплексну методику тестування безпеки. Децентралізовані системи, що використовують низькорівневі мережеві пакети та ручне керування криптографічними примітивами, мають підвищений ризик виникнення специфічних уразливостей, таких як помилки керування пам'яттю, витоки секретних даних через логування або неправильне використання генераторів випадкових чисел. Методика оцінки захищеності проєкту поєднує два взаємодоповнюючі підходи, а саме статичний аналіз вихідного коду (*Static Application Security Testing*) та динамічний аналіз поведінки застосунку під час мережевої взаємодії вузлів у локальному середовищі.

Статичний аналіз вихідного коду мовою *Go* було реалізовано за допомогою спеціалізованого програмного забезпечення для автоматичного визначення вразливостей під назвою *Gosec* [22]. Цей інструмент здійснює глибокий аналіз абстрактного синтаксичного дерева програми, порівнюючи знайдені конструкції з базою відомих безпекових дефектів та криптографічних помилок. Програма автоматичного сканування фокусується на перевірці використання генератора випадкових чисел, контролюючи, щоб усі вектори ініціалізації (*nonce*) та сесійні ефемерні ключі еліптичної кривої *X*25519 створювалися виключно через безпечний системний інтерфейс *crypto*/*rand*. Використання слабкого псевдовипадкового генератора *math*/*rand* було б миттєво зафіксоване сканером як критична помилка.

Але все ж було знайдено декілька помилок в коді програми які були виправлені після сканування (рис. 4.1).

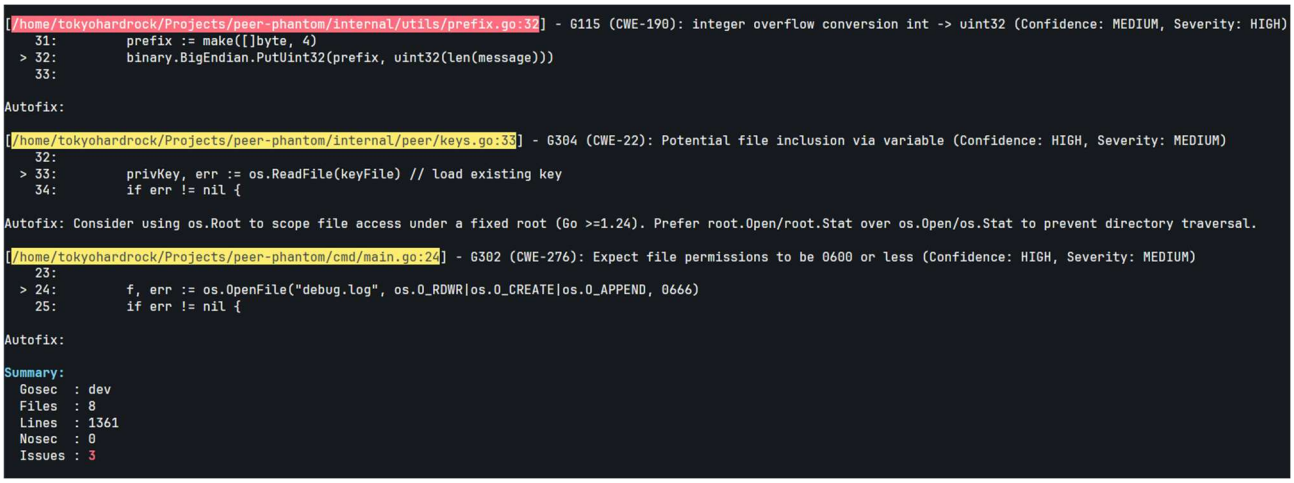

Рисунок 4.1 - Результат роботи статичного аналізатора коду *Gosec*

Виходячи з рисунку 4.1 бачимо, що першим попередженням із кодом *G*115 є потенційне переповнення цілих чисел при приведенні типів (*integer overflow conversion*). Це виникає через приведення типу *int* (який може мати розрядність 32 або 64 біти залежно від розрядності операційної системи) до фіксованого типу *uint*32. Проблему було вирішено шляхом додавання константи, яка відповідає за жорсткий ліміт довжини повідомлення, та впровадження перевірки, чи обсяг вхідних даних не перевищує вказану межу. Оскільки довжина текстових повідомлень у месенджері рідко перевищує 250–300 символів, було встановлено ліміт у 10 Кбайт, чого цілком достатньо для такого об'єму інформації. Крім того, розрядність префікса, який зберігає довжину повідомлення в мережевому потоці, було оптимізовано й зменшено з 32 до 16 біт (тип *uint*16).

Наступне попередження з кодом *G*304 не є критичною вразливістю у контексті даного проєкту. Воно вказує на потенційний ризик атаки типу *Path Traversal*, яка дозволяє зловмиснику шляхом додавання спеціальних символів (наприклад, ../) вийти за межі кореневого каталогу та отримати доступ до конфіденційних системних файлів. У розробленому застосунку цей вектор атаки повністю нівельовано, оскільки шлях до файлу з приватним ключем ідентифікації користувача не вказується оператором вручну, а є статичним і жорстко визначеним у коді – файл створюється автоматично безпосередньо у

робочій директорії разом із виконуваним файлом. Таким чином, можливість підміни шляху з боку зовнішнього порушника повністю відсутня.

Останнє зауваження з кодом *G*302 стосувалося надлишкових прав доступу під час створення файлів застосунку, зокрема файлів логування. Початкові права доступу 0666 дозволяли будь-якому користувачеві в операційній системі зчитувати та модифікувати ці дані. Для усунення цієї вразливості конфігурацію створення файлів було рефакторизовано: права доступу було суворо обмежено до маски 0600. Це гарантує, що лише власник поточного процесу (користувач, який запустив чат) має ексклюзивні права на читання та запис конфіденційної інформації, що повністю відповідає стандартам безпеки розмежування доступу в *UNIX*-подібних системах.

4.2. Аналіз стійкості до атак

Практичний аналіз криптографічної стійкості та захищеності децентралізованого застосунку проводився шляхом симуляції типових векторів атак на однорангові мережі в ізольованому лабораторному середовищі. Основними критеріями оцінки виступали збереження конфіденційності вмісту повідомлень при повному контролі зловмисника над фізичним каналом зв'язку, а також неможливість компрометації автентичності учасників розмови з боку сторонніх осіб. Для підтвердження результатів тестування було використано метод інспекції мережевих пакетів за допомогою програмного аналізатора *Wireshark* [23] та фіксацію логів роботи ядра системи.

Першим етапом випробувань стала практична перевірка захищеності системи від пасивного перехоплення мережевого трафіку (сніффінгу). Для цього на мережевому інтерфейсі хоста було активовано режим перехоплення пакетів, після чого між двома тестовими вузлами було здійснено обмін повідомленнями. На рисунку 4.2 продемонстровано дамп мережевого кадру, вихопленого у момент передачі конфіденційного тексту при вимкненому

шифруванні. На рисунку 4.3 продемонстровано дамп мережевого кадру при увімкненому шифруванні.

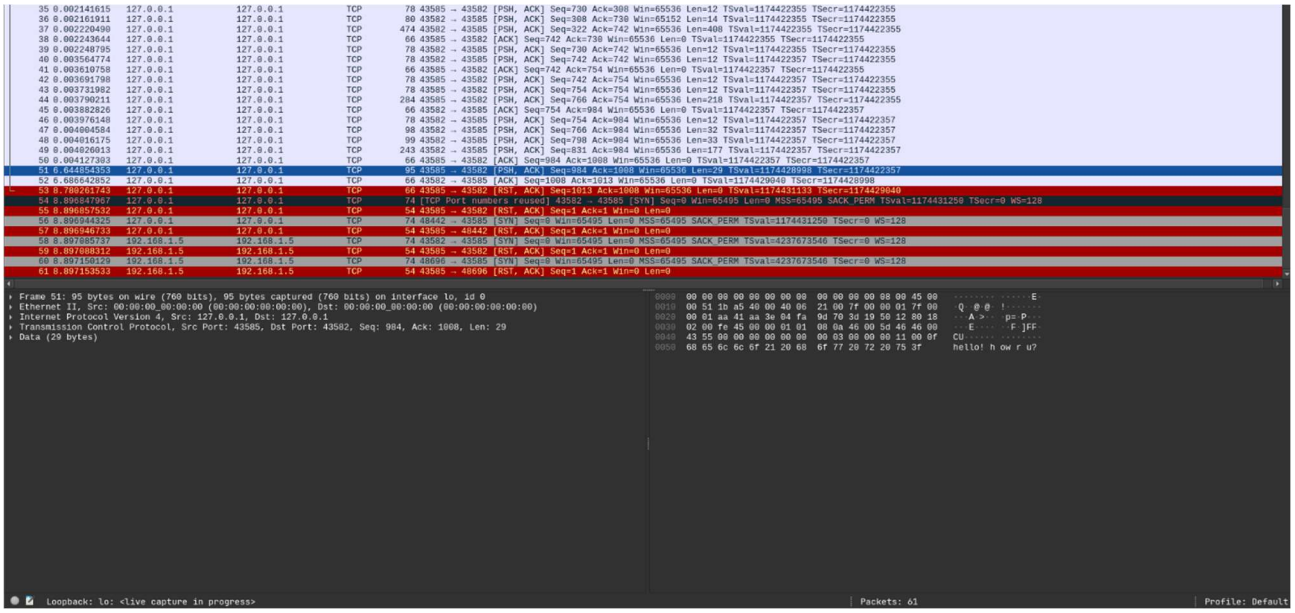

Рисунок 4.2 – Перехоплення відкритого пакету даних у програмі *Wireshark*

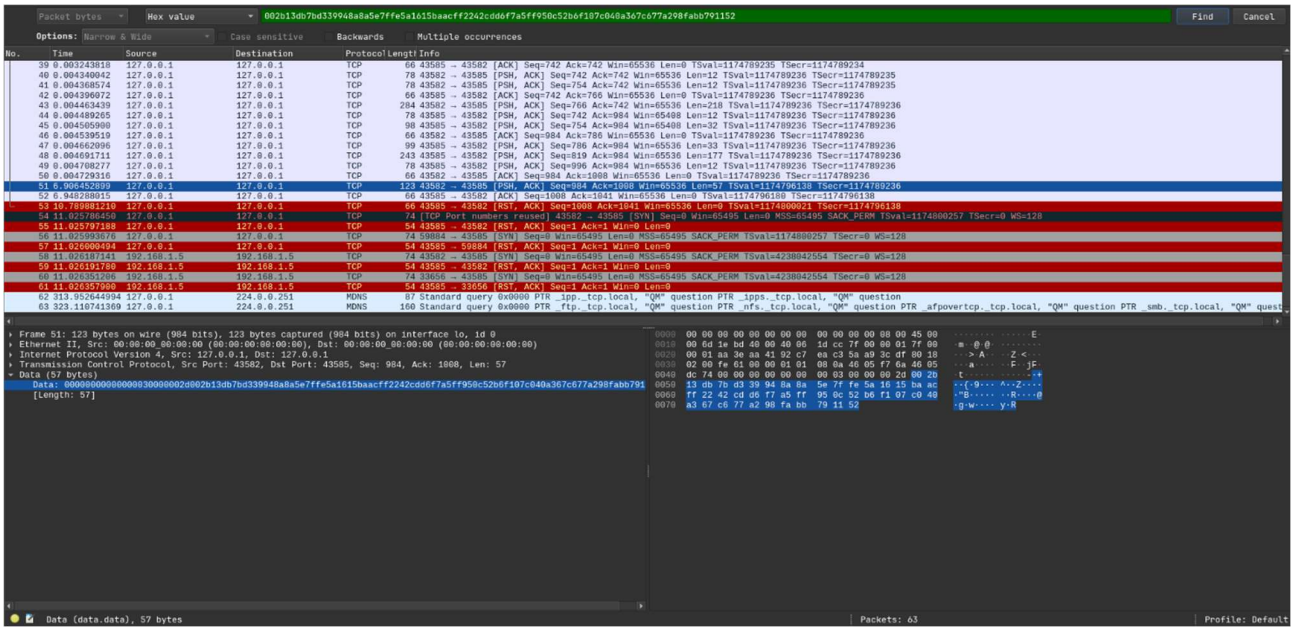

Рисунок 4.3 – Перехоплення зашифрованого пакету даних у програмі *Wireshark*

Аналіз структури перехопленого пакету на рисунку 4.3 чітко доводить, що сторонній спостерігач не здатний отримати доступ до відкритого тексту

повідомлення. У полі даних *TCP*-фрейма повністю відсутні будь-які текстові символи, ідентифікатори користувачів або службові команди у відкритому вигляді. Замість цього пакет містить суцільний масив шістнадцяткових даних, де після початкового префікса довжини слідують 12 байтів випадкового вектора ініціалізації (*nonce*), а далі – бінарний шифротекст із приєднаним автентифікаційним тегом. Оскільки для захисту каналу застосовано симетричний шифр *AES*-256-*GCM* з унікальним ключем для кожної сесії, пасивне декодування перехопленого мережевого шуму є математично неможливим за прийнятний час, що підтверджує абсолютну стійкість чату до атак прослуховування.

Другим, більш складним етапом стала практична перевірка стійкості децентралізованого застосунку до активного деструктивного впливу, а саме до атаки типу «людина посередині» (*Man-in-the-Middle*, *MITM*) на етапі криптографічного рукостискання. Оскільки в поточній конфігурації транспорту навмисно було активовано режим без вбудованої безпеки каналу, базовий захист покладався виключно на логіку автентифікованого наскрізного шифрування (*E2EE*), реалізовану безпосередньо на рівні додатку. Головною метою цього етапу тестування було підтвердження здатності розробленого протоколу протидіяти спробам зловмисника динамічно підмінити відкриті ключі легітимних вузлів та перехопити керування сесією.

Для реалізації цієї загрози було спроектовано та програмно реалізовано імітаційну модель перехоплювача – спеціалізований *TCP*-проксі сервер на мові програмування *Go*. Цей інструмент розгортався як проміжний маршрутизатор трафіку, що перехоплював з'єднання від ініціатора сесії, дуплексно перенаправляв байтовий потік кінцевому одержувачу та здійснював автоматичний аналіз структури пакетів. Алгоритм аналізу мережевих пакетів, інтегрований у структуру *MITM*-проксі, був націлений на виявлення серіалізованих *JSON*-структур обміну ключами, які містять службові поля відкритого ефемерного ключа та його цифрового підпису.

У ході симуляції атаки *MITM*-вузол успішно виявив пакет криптографічного рукостискання. З метою компрометації конфіденційності подальшої розмови зловмисник здійснив спробу нав'язати власні параметри шифрування шляхом заміни оригінального відкритого сесійного ключа *X*25519 на власне згенероване значення. Модифікація даних виконувалася шляхом зміни випадкового символу ключа на символ *Q*, що дозволило змінити безпосередньо криптографічний вміст, зберігши при цьому загальну синтаксичну цілісність структури для успішного парсингу приймаючою стороною. Нижче наведений вихідний код проксі-сервера для перехоплення:

```
package main

import (
	"bytes"
	"fmt"
	"io"
	"log"
	"net"
)
const (
	FAKE_ADDR = "127.0.0.1:9999"
	REAL_ADDR = "127.0.0.1:4001"
)
func main() {
	listener, err := net.Listen("tcp", FAKE_ADDR)
	if err != nil {
		log.Fatalf("Error during running tcp server: %v", err)
	}
	defer listener.Close()
	fmt.Printf("Listen on %s\n", FAKE_ADDR)
```

```
        fmt.Printf("Redirect to %s\n\n", REAL_ADDR)
        for {
                inConn, err := listener.Accept()
                if err != nil {
                        log.Printf("Incoming connection error: %v", err)
                        continue
                }
                log.Printf("Incoming connection from: %s\n",
inConn.RemoteAddr().String())
                outConn, err := net.Dial("tcp", REAL_ADDR)
                if err != nil {
                        log.Printf("Unable to dial to %s: %v", REAL_ADDR, err)
                        inConn.Close()

                        continue
                }
                go proxyTraffic(inConn, outConn)
                go proxyTraffic(outConn, inConn)
        }
}
func proxyTraffic(src, dst net.Conn) {
        defer src.Close()
        defer dst.Close()

        buf := make([]byte, 4096)

        for {
                n, err := src.Read(buf)
                if err != nil {
                        if err != io.EOF {
```

```
                log.Printf("Read error: %v", err)
            }
            return
        }
        payload := buf[:n]
        if bytes.Contains(payload, []byte(`"signedPubKey"`)) {
            fmt.Println("Received handshake frame")
            fmt.Printf("Raw: %s\n", string(payload))

            if bytes.Contains(payload, []byte(`"pubKey"`)) {
                fmt.Println("Modifying key")

                idx := bytes.Index(payload, []byte(`"pubKey"`))
                if idx != -1 && idx+15 < len(payload) {
                    payload[idx+15] = 'Q'
                }
                fmt.Println("Modified key:")
                fmt.Printf("Raw: %s\n", string(payload))
            }
        }
        _, err = dst.Write(payload)
        if err != nil {
            log.Printf("Write error: %v", err)
            return
        }
    }
}
```

Результати фіксації логів роботи ядра системи у момент проведення активного втручання показали бездоганну ефективність розробленого

криптографічного бар'єра. Після отримання модифікованого пакета рукостискання вузол-ініціатор автоматично запустив процедуру верифікації автентичності відправника. Ключовою перевагою розробленої архітектури є те, що відкритий ключ для функції перевірки підпису отримувався не з мережевого пакета, який міг бути підроблений, а безпосередньо з унікального ідентифікатора транспортного з'єднання, жорстко прив'язаного до криптографічного хешу довгострокового ключа *Ed*25519 легітимного співрозмовника.

Оскільки зловмисник не володів приватним ключем легітимного вузла, він не мав математичної можливості згенерувати валідний цифровий підпис під зміненими байтами сесійного ключа. На рисунку 4.4 та 4.5 продемонстровано лог консолі приймаючого вузла, який зафіксував факт деструктивного втручання в канал зв'язку та лог нашого проксі-сервера.

Рисунок 4.4 – Лог відхилення модифікованого пакета та розриву сесії вузлом-отримувачем

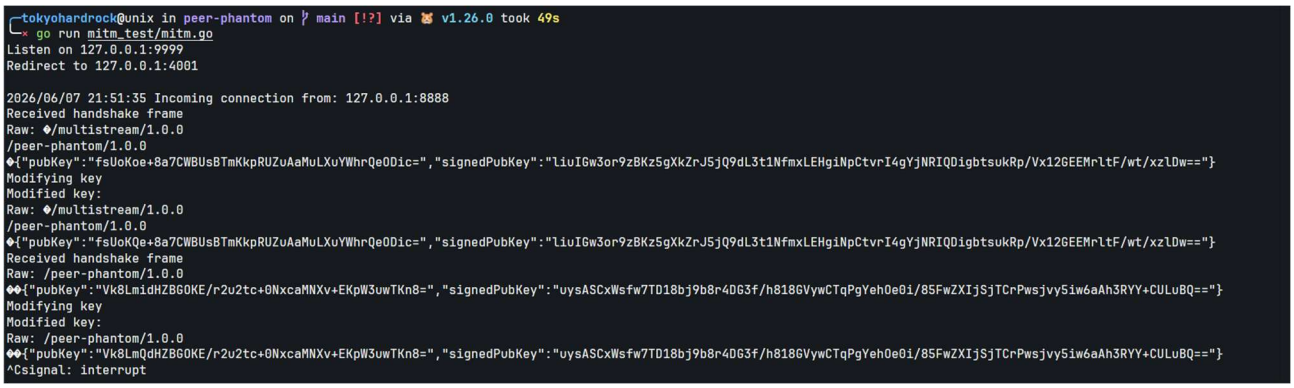

Рисунок 4.4 – Лог роботи проксі-сервера, що модифікує дані

Як свідчать результати тестування на рисунку 4.5, математичний апарат алгоритму цифрового підпису *Ed*25519 миттєво виявив факт порушення

цілісності криптографічних даних. Вбудована функція верифікації повернула критичну помилку, внаслідок чого застосунок негайно розірвав транспортну *TCP*-сесію в односторонньому порядку, повністю заблокувавши подальший обмін даними ще до моменту відправки першого текстового повідомлення.

Таким чином, експериментальні випробування активних методів зламу підтвердили високий рівень захищеності розробленого децентралізованого застосунку. Використання механізму криптографічної прив'язки до каналу зв'язку (*Channel Binding*) на основі ідентифікаторів *PeerID* у поєднанні з автентифікованим протоколом Діффі-Гіллмана повністю нівелює загрозу *MITM*-атак, забезпечуючи автентичність учасників та цілісність ключів у неконтрольованому мережевому середовищі.

Висновки до розділу

У четвертому розділі виконано експериментальну оцінку захищеності та ефективності розробленого P2P-чату методами статичного й динамічного аналізу. Перевірка вихідного коду інструментом Gosec дозволила виявити та усунути критичні вразливості, пов’язані з переповненням цілих чисел і надлишковими правами доступу до файлів. Аналіз мережевого трафіку у Wireshark підтвердив стійкість системи до пасивного перехоплення даних завдяки повному шифруванню передаваних повідомлень. Додатково експеримент із використанням шкідливого TCP-проксі продемонстрував ефективний захист від MITM-атак: будь-яка спроба підміни ключів під час рукостискання виявлялася механізмом цифрових підписів Ed25519 і призводила до автоматичного розриву з’єднання до початку обміну повідомленнями.

ВИСНОВОК

У дипломній роботі вирішено задачу проєктування, розробки та аналізу захищеності децентралізованої системи обміну повідомленнями з наскрізним шифруванням. На основі аналізу сучасних підходів до захищених комунікацій обґрунтовано доцільність використання *P2P*-архітектури без проміжних серверів, що дозволяє усунути ризики, пов’язані з централізованим зберіганням даних. Для реалізації системи обрано мову програмування Go, яка забезпечує високу продуктивність і ефективну роботу в асинхронному середовищі.

Для забезпечення безпеки розроблено гібридну криптографічну схему на основі *ECDH* (*Curve*25519), *Ed*25519, *HKDF-SHA*256 та *AES*-256-*GCM*, яка гарантує конфіденційність, цілісність, автентичність повідомлень і захист від атак типу «людина посередині». Програмна архітектура побудована за подієво-орієнтованим принципом із повною ізоляцією інтерфейсного, мережевого та криптографічного рівнів, а користувацький інтерфейс реалізовано у вигляді кросплатформенного текстового консольного клієнта з низькими вимогами до ресурсів.

Результати тестування підтвердили високу стійкість системи до поширених загроз. Статичний аналіз коду дозволив виявити та усунути потенційні вразливості, а дослідження мережевого трафіку в *Wireshark* засвідчило відсутність витоку відкритих даних. Експерименти з моделюванням *MITM*-атак показали, що будь-які спроби підміни криптографічних параметрів виявляються механізмом цифрових підписів і призводять до автоматичного розриву з’єднання. Отримані результати підтверджують, що розроблений месенджер відповідає вимогам безпеки та може використовуватися як основа для автономних систем захищеного обміну інформацією.

ПЕРЕЛІК ПОСИЛАНЬ

1. ДСТУ *ISO/IEC* 27001:2015 Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги (*ISO/IEC* 27001:2013, *IDT*). – К. : ДП «УкрНДНЦ», 2016. – 32 с.

2. *Steinmetz R., Wehrle K. Peer-to-Peer Systems and Applications. – Berlin: Springer Science & Business Media*, 2005. – 629 *p*.

3. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = *Applied Cryptography. Protocols, Algorithms, and Source Code in C*. – М. : Триумф, 2016. – 816 с.

4. *Man-in-the-Middle (MITM) Attack: Types, Techniques and Prevention* [Електронний ресурс]. – Режим доступу: *https://beaglesecurity.com/blog/article/man-in-the-middle-attack.html*.

5. *Signal Protocol* [Електронний ресурс]. – Режим доступу: *https://en.wikipedia.org/wiki/Signal_Protocol*.

6. *Matrix Protocol* [Електронний ресурс]. – Режим доступу: https://matrix.org/foundation/about/.

7. *Matrix End-to-End Encryption implementation guide* [Електронний ресурс]. – Режим доступу: *https://matrix.org/docs/matrix-concepts/end-to-end-encryption/#implementing-end-to-end-encryption-in-matrix-clients*.

8. Maymounkov P., Eres D. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. Lecture Notes in Computer Science [Електронний ресурс]. – Режим доступу: *https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf*

9. *Bernstein D. J. Curve*25519: *new Diffie-Hellman speed records* // *Public Key Cryptography – PKC* 2006. *Lecture Notes in Computer Science*. – 2006. – *Vol*. 3958. – *P*. 207–228.

10. *Briar: How it works* [Електронний ресурс]. – Режим доступу: *https://briarproject.org/how-it-works/*

11. What is end-to-end encryption (E2EE)? [Електронний ресурс]. – Режим доступу: *https://www.ibm.com/think/topics/end-to-end-encryption*

12. *M. A. Al-Shabi. A Survey on Symmetric and Asymmetric Cryptography Algorithms in information Security // International Journal of Scientific and Research Publications. – Vol.* 9, *Issue* 3. – *March* 2019.

13. NAT traversal в embedded P2P-мессенджере на Go: почему overlay routing, а не STUN/TURN/ICE [Електронний ресурс]. – Режим доступу: *https://habr.com/ru/articles/1043134/*

14. *RFC* 7748 *Elliptic Curves for Security / A. Langley, M. Hamburg, S. Turner. – Internet Engineering Task Force* (*IETF*), 2016. – 18 *p.*

15. *RFC* 5084 *Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax* (*CMS*) / *R. Housley. – Internet Engineering Task Force* (*IETF*), 2007. – 11 *p.*

16. *McGrew D. A., Viega J. The Galois/Counter Mode of Operation* (*GCM*) // *Submission to NIST Modes of Operation Process*. – 2004. – *Vol*. 20. – *P*. 1–44.

17. *Y. Nir, A. Langley ChaCha20 and Poly1305 for IETF Protocols – RFC* 8439, *IRTF*, 2018.

18. *H. Krawczyk HMAC-based Extract-and-Expand Key Derivation Function (HKDF) – RFC* 5869, *IETF*, 2010.

19. NaCl (Networking and Cryptography library) [Електронний ресурс] – Режим доступу: *https://nacl.cr.yp.to/*

20. *Libp2p* [Електронний ресурс] – Режим доступу до ресурсу: *https://libp2p.io/*.

21. Програмна реалізація механізмів наскрізного шифрування в *P2P*-чаті «*Peer-Phantom*» [Електронний ресурс] / репозиторій вихідного коду – Режим доступу: *https://github.com/tokyohardrock/peer-phantom*.

22. *Gosec (Go security checker)* [Електронний ресурс]. – Режим доступу: *https://github.com/securego/gosec*

23. *Wireshark User's Guide* [Електронний ресурс]. – Режим доступу: *https://www.wireshark.org/docs/wsug_html_chunked/*