Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

ЧАТ ДЛЯ ЛОКАЛЬНОЇ МЕРЕЖІ З ВИКОРИСТАННЯМ СОКЕТІВ І ТЕХНОЛОГІЇ ШИФРУВАННЯ ПОВІДОМЛЕНЬ

Виконав:
здобувач IV курсу
групи ІПЗ-41
спеціальності 121 «Інженерія програмного забезпечення»
Рак Дмитро Олександрович

Науковий керівник:
к.п.н., доцент Петренко С. В.

Рівне – 2026

ЗМІСТ

ВСТУП

Актуальність дослідження. Питання забезпечення конфіденційності та цілісності інформації під час її передачі мережевими каналами залишається ключовою проблемою розробки комунікаційних систем. Використання публічних хмарних сервісів для обміну повідомленнями створює додаткові вектори атак та ризики витоку чутливих даних. Побудова ізольованих локальних мереж з власним сервером дозволяє мінімізувати залежність від зовнішніх провайдерів, проте вимагає впровадження надійних механізмів криптографічного захисту на всіх етапах обробки інформації.

Новітні тенденції у сфері мережевої безпеки демонструють перехід від виключно транспортного шифрування до комплексного підходу, який обов'язково включає захист збережених даних. Зберігання історії листування у відкритому вигляді на серверах баз даних становить критичну вразливість. Застосування підходів транспортного захисту на основі протоколів SSL/TLS у поєднанні з криптографічним перетворенням змісту повідомлень безпосередньо на рівні бази даних гарантує безпеку навіть у випадку компрометації фізичного носія або отримання несанкціонованого доступу до системи управління даними..

Мета дослідження полягає у розробці кросплатформної клієнт-серверної системи обміну текстовими повідомленнями для забезпечення конфіденційності комунікацій у локальних комп'ютерних мережах. Дослідження спрямоване на створення програмного комплексу із застосуванням технологій SSL/TLS та шифрування баз даних задля запобігання несанкціонованому доступу до історії листування та перехопленню мережевого трафіку у контексті побудови автономних та захищених систем зв'язку.

Відповідно до поставленої мети сформульовано перелік **завдань**:

1. Дослідити існуючі теоретичні підходи та програмні рішення у сфері криптографічного захисту мережевого трафіку і збережених даних;

2. Спроєктувати архітектуру захищеної клієнт-серверної системи, базу даних та логіку мережевої взаємодії;

3. Розробити алгоритмічне забезпечення обробки клієнтських запитів, парсингу команд та криптографічних протоколів;

4. Імплементувати серверну та клієнтську частини програмного комплексу із застосуванням мови програмування C++ та відповідних бібліотек;

5. Протестувати створену систему на предмет стабільності, цілісності передачі даних та коректності роботи механізмів безпеки.

Об’єктом дослідження є процес захищеного обміну текстовою інформацією та управління обліковими даними користувачів у локальних комп’ютерних мережах. Дослідження спрямоване на вивчення механізмів криптографічного перетворення та маршрутизації мережевих пакетів з метою мінімізації ризиків компрометації конфіденційних повідомлень під час їх передачі та тривалого зберігання.

Предметом дослідження є алгоритми, протоколи та програмні засоби забезпечення конфіденційності передачі даних за протоколом SSL/TLS і захищеного зберігання історії листування на базі симетричного шифрування. Дослідження спрямоване на оптимізацію архітектури клієнт-серверної взаємодії для побудови стійкого до перехоплення та несанкціонованого доступу локального чату.

Методи дослідження

До теоретичних методів належать аналіз наукової та технічної літератури для вивчення стану проблеми, а також порівняння існуючих протоколів шифрування та архітектурних рішень для обґрунтування технологічного стеку. Абстрагування та концептуальне моделювання застосовувалися під час розробки логічної моделі захищеного обміну даними та формування вимог до архітектури.

Емпіричні методи включають проєктування логіки взаємодії клієнта та сервера, безпосередню розробку програмного забезпечення із застосуванням мови

програмування C++ та відповідного фреймворку, а також практичне тестування з’єднань на базі мережевих сокетів.

Практичне значення одержаних результатів полягає у створенні готового до розгортання програмного продукту, який забезпечує надійний та ізольований обмін конфіденційною інформацією. Розроблений програмний комплекс орієнтований на впровадження у корпоративних локальних мережах підприємств, державних установ або закритих організацій, де критично важливою вимогою є збереження корпоративної таємниці без виходу у глобальну мережу та без використання сторонніх серверів.

Застосований підхід до комплексного шифрування мережевого трафіку та збережених у базі даних повідомлень гарантує високий рівень безпеки навіть у разі компрометації фізичного доступу до серверного обладнання. Наявність вбудованої консолі адміністрування та використання багатоплатформних технологій дають змогу гнучко керувати правами доступу, масштабувати систему та адаптувати її під специфічні вимоги безпеки конкретної установи.

Апробації і впровадження результатів дослідження. Основні теоретичні результати дослідження обговорювалися на Міжнародній науковій конференції «Сучасні дослідження та інноваційні підходи в науці» (м. Ноттінгем, 23.01.2026) [1] та на І Міжнародній науково-практичній конференції «Сучасні інформаційні технології: від теорії до практики» (м. Луцьк, Луцький національний технічний університет, 22–23 травня 2026) [34].

Структура роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків та списку використаних джерел. Загальний обсяг роботи становить 50 сторінок. Список використаних джерел включає 34 найменувань.

Додано примітку [1]: додати Луцьк

Додано примітку [2]: не впевнений як правильно його додати, тому написав через "та", а ще як їх правильно оформити в посиланнях?

Додано примітку [3]: Рак Д. Порівняльний аналіз сучасних протоколів шифрування та їх вплив на безпеку даних Д. Рак, С. Петренко // І Міжнародна науково-практична конференція
Сучасні інформаційні технології: від теорії до практики – Луцьк : Луцький національний технічний університет, 2026. – С. ХХ–ХХ.

Додано примітку [4]: якщо я правильно зрозумів, тоді так

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СТАНУ ПРОБЛЕМИ ЗАХИЩЕНОГО ОБМІНУ ДАНИМИ

1.1. Еволюція та сучасний стан систем корпоративного обміну повідомленнями

Історичний розвиток систем корпоративного обміну повідомленнями тісно пов’язаний з еволюцією комп’ютерних мереж та архітектурних підходів до обробки даних. На ранніх етапах розгортання локальних обчислювальних мереж комунікація між вузлами базувалася на простих широкомовних протоколах, які не передбачали складних механізмів автентифікації чи шифрування трафіку. Локальна мережа розглядалася як фізично захищений периметр — такий підхід призводив до повного ігнорування внутрішніх інсайдерських загроз. Згодом відбувся перехід до класичної архітектури взаємодії клієнта та сервера, яка дозволила централізувати управління повідомленнями, проте питання криптографічного захисту залишалося другорядним через обмежені обчислювальні ресурси кінцевих станцій.

Масштабування бізнесу та розширення географії корпоративної взаємодії зумовили масову міграцію компаній на глобальні хмарні платформи обміну повідомленнями. Відповідні рішення забезпечують високу доступність та зручність використання, легко інтегруючись у різноманітні операційні системи. Проте перехід до моделі програмного забезпечення як послуги кардинально змінив парадигму мережевої безпеки. Контроль над фізичною та логічною інфраструктурою повністю перейшов до сторонніх провайдерів — це створило нові вектори атак на корпоративну інформацію.

Використання популярних публічних месенджерів у закритому корпоративному середовищі генерує критичні ризики порушення конфіденційності

даних. Основна проблема полягає у відсутності прозорості алгоритмів обробки інформації на стороні хмарного сервера та неможливості гарантувати безповоротне видалення чутливих повідомлень з баз даних провайдера. Аналіз мережевих загроз демонструє, що делегування управління криптографічними ключами стороннім сервісам нівелює ефективність транспортного шифрування, оскільки дані залишаються вразливими у стані спокою на серверах постачальника послуг [2, 3].

Проблематика захисту корпоративних комунікацій та моделювання мережевих загроз активно досліджується фахівцями у галузі кібербезпеки. Дослідження В. Сталлінгса та Б. Шнайєра доводять необхідність впровадження криптографічних протоколів з наскрізним контролем ключів безпосередньо на рівні автономної інфраструктури організації [2, 3]. Роботи вітчизняних науковців, зокрема І. Д. Горбенка, підтверджують, що побудова захищеного середовища вимагає ізоляції комунікаційних каналів від глобальної мережі та застосування локальних серверних рішень, здатних забезпечити комплексне криптографічне перетворення транзитного трафіку і збережених баз даних [4].

Проєктування інфраструктури систем корпоративного обміну повідомленнями передбачає суворе дотримання архітектурних принципів автономності та ізольованості обчислювального середовища. В умовах закритих локальних мереж оптимальною є класична клієнт-серверна модель, яка дозволяє централізувати управління обліковими записами та забезпечити єдину точку контролю над інформаційними потоками. На відміну від децентралізованих одноранговых мереж, наявність виділеного сервера гарантує цілісність збереження історії листування та синхронізацію стану клієнтських додатків незалежно від часу їхнього підключення до комунікаційного каналу. Застосування єдиного центру обробки запитів спрощує маршрутизацію мережевих пакетів всередині ізольованої інфраструктури.

Функціонування комунікаційних систем у корпоративному сегменті суворо регламентується міжнародними та галузевими стандартами інформаційної безпеки.

Вимоги стандартів серії ISO/IEC 27000 та спеціальних публікацій Національного інституту стандартів і технологій США, зокрема документа NIST SP 800-53, визначають необхідність застосування багаторівневого захисту інформаційних ресурсів [5]. Згідно з цими нормативними документами, локальний програмний комплекс повинен підтримувати надійну автентифікацію користувачів, жорстке розмежування прав доступу на рівні віртуальних кімнат та безперервне логування системних подій. Відповідність вказаним критеріям вимагає імплементації дієвих механізмів контролю цілісності під час розгортання клієнтських та серверних компонентів [6].

Фізична ізоляція локальної мережі від зовнішньої глобальної інфраструктури не здатна повністю виключити ймовірність несанкціонованого доступу до конфіденційних даних. Аналіз векторів атак доводить значну частку внутрішніх загроз, пов’язаних із цілеспрямованими діями інсайдерів або компрометацією окремих кінцевих вузлів. У випадку успішного перехоплення транзитного мережевого трафіку всередині корпоративного периметра третя сторона отримує можливість читання або модифікації повідомлень, які передаються у відкритому вигляді. Наявність потенційних вразливостей у мережевому комутаційному обладнанні підтверджує недостатність виключно апаратного екранування локальної мережі [7].

Нейтралізація описаних внутрішніх загроз створює безальтернативну необхідність використання надійних криптографічних протоколів на всіх стадіях обробки інформації. Безпечна клієнт-серверна взаємодія вимагає інтеграції стандартизованих рішень транспортного рівня для забезпечення взаємної автентифікації вузлів та симетричного шифрування сеансів зв'язку. Водночас захист повідомлень у стані спокою на боці центрального сервера потребує криптографічного перетворення даних безпосередньо у реляційних таблицях. Поєднання транспортного захисту та стійкого шифрування баз даних формує

технологічний фундамент для розробки сучасних захищених локальних систем зв’язку [8].

1.2. Аналіз теоретичних підходів до криптографічного захисту інформації

Теоретичний фундамент криптографічного захисту інформації в мережах передачі даних базується на принципах забезпечення конфіденційності, цілісності та автентичності повідомлень. Захист транспортного рівня розглядається як первинний рубіж оборони від атак типу людина посередині та пасивного перехоплення трафіку. Математичні моделі стійкості криптографічних систем доводять, що надійність шифрування має спиратися виключно на секретність ключа, а не на прихованість самого алгоритму. Відповідний концептуальний підхід є базовим для проєктування відкритих мережевих протоколів, де механізми перетворення даних проходять незалежний криптоаналіз. Дослідження Е. Рескорли та Д. Боне підтверджують необхідність застосування гібридних криптосистем для досягнення оптимального балансу між криптографічною стійкістю та продуктивністю комунікаційних вузлів [9, 10].

Практична реалізація транспортного захисту спирається на стандарти родини SSL/TLS, архітектура яких регламентується відповідними документами інженерної ради інтернету, зокрема RFC 8446 для версії TLS 1.3. Встановлення захищеного з’єднання ініціюється процедурою криптографічного рукостискання, що використовує алгоритми асиметричної криптографії для взаємної автентифікації та безпечного обміну сесійними ключами. Класичним теоретичним рішенням для цієї задачі є алгоритм RSA, розроблений Р. Рівестом, А. Шаміром та Л. Адлеманом. Стійкість RSA базується на обчислювальній складності задачі факторизації великих цілих чисел. Використання пари математично пов'язаних ключів — відкритого для шифрування та закритого для розшифрування — дозволяє клієнту

безпечно передати серверу параметри для генерації спільного секрету навіть через незахищений канал зв’язку [11].

Через високу ресурсоємність математичних операцій з великими простими числами асиметрична криптографія застосовується виключно на етапі ініціалізації з’єднання. Після успішного обміну ключовим матеріалом протокол TLS переходить до використання симетричних алгоритмів шифрування для захисту основного потоку даних. Такий гібридний підхід мінімізує затримки під час передачі великих обсягів інформації. Забезпечення цілісності кожного переданого блоку даних реалізується шляхом додавання кодів автентифікації повідомлень, що унеможливлює непомітну модифікацію чи підробку транзитних пакетів зловмисником [6].

Окрім шифрування каналів зв’язку, фундаментальним аспектом інформаційної безпеки є захист автентифікаційних даних користувачів у стані спокою. Теоретичною основою для вирішення цієї проблеми є використання криптографічних функцій гешування. За математичною природою така функція є одностороннім перетворенням масиву даних довільної довжини у бітову послідовність фіксованого розміру. Критичними вимогами до стійких алгоритмів є незворотність перетворення, стійкість до колізій першого та другого роду, а також лавинний ефект — кардинальна зміна результуючого значення навіть за мінімальної зміни вхідного повідомлення. Зберігання паролів у вигляді хешованих значень виключає можливість відновлення оригінального пароля навіть у разі отримання зловмисником повного доступу до бази даних [6, 12].

Сучасним галузевим стандартом у сфері криптографічного гешування є алгоритм SHA-256, що входить до родини SHA-2 та регламентується специфікацією NIST FIPS 180-4. Алгоритм оперує тридцятидвохрозрядними словами та виконує шістдесят чотири раунди логічних перетворень, формуючи на виході двістіп’ятдесятшестирозрядний дайджест. Для протидії атакам на основі попередньо обчислених словників та райдужних таблиць теоретичні моделі

безпеки вимагають обов’язкового додавання криптографічної солі — унікальної випадкової послідовності байтів — до кожного пароля перед його обробкою. Праці Х. Кравчика щодо механізмів виведення ключів доводять, що комбінація алгоритму SHA-256 із застосуванням унікальної солі забезпечує надійний захист облікових даних від методів криптоаналізу та прямого перебору [12, 13].

Забезпечення конфіденційності інформації вимагає комплексного підходу, який не обмежується лише захистом каналів зв'язку. Фундаментальною концепцією інформаційної безпеки є захист даних у стані спокою, що передбачає криптографічне перетворення інформації, яка фізично зберігається на цифрових носіях або у реляційних базах даних. Теоретичні моделі загроз вказують на критичну вразливість серверів баз даних до атак, пов’язаних із фізичним викраденням накопичувачів, несанкціонованим створенням резервних копій або ескалацією привілеїв адміністраторами інфраструктури. Для мінімізації цих ризиків необхідно впроваджувати механізми шифрування безпосередньо на рівні системи управління базами даних [14].

Теоретичною основою для реалізації захищеного зберігання повідомлень є алгоритми симетричного шифрування, зокрема розширений стандарт шифрування AES. На відміну від транспортного рівня, де ключі генеруються динамічно для кожної сесії, шифрування баз даних вимагає управління довгостроковими криптографічними ключами. Математично обґрунтованим підходом є шифрування на рівні окремих стовпців або комірок таблиць, що дозволяє вибірково захищати чутливу інформацію, наприклад, зміст текстових повідомлень, залишаючи метадані доступними для швидкого пошуку. Використання режимів зчеплення блоків шифротексту або лічильника з автентифікацією Галуа гарантує високу стійкість до криптоаналізу. Практичним втіленням цих теоретичних концепцій є використання спеціалізованих криптографічних розширень для реляційних систем, які виконують перетворення даних безпосередньо під час виконання структурованих запитів [15, 16].

Стратегія глибокого ешелонування захисту вимагає чіткого розділення транспортної безпеки та безпеки зберігання. Аналіз архітектурних патернів кібербезпеки доводить, що застосування виключно протоколів SSL/TLS захищає інформацію лише під час транзиту через мережеве середовище. Після розшифрування мережевих пакетів на сервері дані потрапляють до оперативної пам’яті та записуються на дискові накопичувачі у відкритому вигляді. Проєктування стійкої комунікаційної системи передбачає створення незалежних контурів шифрування. Перехоплення або компрометація сесійних ключів транспортного протоколу не повинна призводити до розкриття архіву листування, захищеного довгостроковими ключами симетричного шифрування бази даних [17].

Проведений теоретичний аналіз криптографічних підходів формує наукове підґрунтя для розробки програмного комплексу захищеного обміну даними. Комбінація гібридного транспортного шифрування для взаємної автентифікації клієнта та сервера, використання алгоритмів криптографічного гешування для захисту облікових записів та симетричного шифрування реляційних баз даних дозволяє побудувати надійну архітектуру. Такий багаторівневий підхід унеможливлює перехоплення комунікацій у локальній мережі та гарантує конфіденційність історії листування навіть за умови фізичної компрометації серверного обладнання.

1.3. Огляд існуючих програмних рішень та обґрунтування напряму дослідження

Ринок програмного забезпечення для організації корпоративного обміну повідомленнями пропонує широкий спектр готових рішень, які базуються на різних архітектурних парадигмах. Історично першим стандартизованим підходом до побудови децентралізованих мереж обміну повідомленнями став протокол XMPP, відомий також як Jabber. Архітектура рішень на базі XMPP передбачає

використання розширюваної мови розмітки XML для передачі даних між клієнтом та сервером у режимі реального часу. Перевагою таких систем є високий ступінь стандартизації, наявність великої кількості відкритих серверних реалізацій та підтримка федерації між різними вузлами. Проте надмірність формату XML генерує значний додатковий мережевий трафік, а підтримка застарілих розширень ускладнює розробку сучасних клієнтських додатків з апаратним прискоренням криптографії.

Наступним етапом розвитку комунікаційного програмного забезпечення є перехід до комплексних платформ спільної роботи, серед яких найбільш поширеними рішеннями для автономного розгортання є Mattermost та Matrix. Платформа Mattermost базується на класичній архітектурі взаємодії з використанням вебтехнологій та REST API, пропонуючи розширений функціонал управління робочими процесами. Протокол Matrix забезпечує децентралізовану архітектуру з можливістю реплікації історії повідомлень між усіма серверами, які беруть участь у роботі віртуальної кімнати. Зазначені рішення гарантують високий рівень відмовостійкості, підтримку багатих форматів медіаданих та гнучкість налаштувань.

Аналіз архітектури сучасних комплексних месенджерів виявляє їхню глибоку залежність від повноцінного вебстеку технологій. Розгортання серверної частини таких систем вимагає наявності потужного апаратного забезпечення для підтримки паралельної роботи вебсерверів, брокерів повідомлень та складних систем кешування баз даних. Клієнтські додатки переважно створюються на базі важких фреймворків типу Electron, що неминуче призводить до надмірного споживання оперативної пам'яті та процесорного часу на робочих станціях. Використання протоколів прикладного рівня, зокрема HTTP, для обміну повідомленнями створює додаткові накладні витрати під час встановлення з'єднання порівняно з використанням нативних інструментів операційної системи.

У контексті розгортання в невеликих, суворо ізольованих комп'ютерних мережах використання розглянутих комплексних платформ є технічно та економічно нераціональним. Надмірна ресурсоємність інфраструктури та багаторівнева складність процесу адміністрування створюють невиправдане навантаження на мережеве обладнання. Впровадження систем рівня Mattermost вимагає конфігурації великого масиву параметрів, більшість з яких орієнтована на інтеграцію із зовнішніми сервісами або на взаємодію у відкритому середовищі, що прямо суперечить концепції закритого локального периметра. Наявність великої кількості надлишкового програмного коду розширює площу можливих кібератак, вимагаючи безперервного моніторингу вразливостей.

Виявлені недоліки підтверджують потребу у розробці спеціалізованих програмних рішень, оптимізованих для роботи виключно у закритих мережах без виходу до інтернету. Використання нативних мережевих сокетів у поєднанні з оптимізованим форматом передачі даних JSON дозволяє кардинально знизити навантаження на апаратні ресурси комутаційного обладнання. Відмова від багатошарового вебстеку на користь прямої взаємодії через захищені сокетні з'єднання забезпечує мінімальні затримки передачі інформації та суттєво спрощує імплементацію строгих криптографічних алгоритмів. Такий архітектурний підхід дозволяє створити стабільну та передбачувану систему, що надійно функціонує в межах обмеженої інфраструктури підприємства.

Виявлені недоліки існуючих комунікаційних платформ формують об'єктивну потребу у перегляді архітектурних підходів до створення систем обміну повідомленнями для ізольованих мереж. Надмірна ресурсоємність та залежність від багаторівневих вебтехнологій роблять використання популярних корпоративних месенджерів неефективним в умовах обмеженої апаратної інфраструктури. Оптимальним вектором розвитку є відмова від універсальних комерційних продуктів на користь розробки спеціалізованих легковажних рішень. Такий підхід вимагає фокусування на мінімізації програмного коду, зниженні

накладних витрат на обробку мережевих пакетів та спрощенні процесів розгортання і подальшого адміністрування системи.

Фундаментом для побудови ефективного локального месенджера є використання низькорівневої мережевої взаємодії на базі сокетів. На відміну від застосування протоколів прикладного рівня, пряма робота з протоколом керування передачею гарантує мінімальні затримки маршрутизації та відсутність надлишкового службового трафіку. Архітектура, побудована на постійних сокетних з'єднаннях, забезпечує миттєву доставку повідомлень та дозволяє реалізувати власні оптимізовані механізми серіалізації даних. Відсутність необхідності розгортання проміжних вебсерверів чи брокерів повідомлень кардинально підвищує загальну надійність програмного комплексу та робить його стійким до пікових навантажень у межах локальної мережі.

Критичною вимогою до сучасного комунікаційного програмного забезпечення є підтримка кросплатформеності без втрати продуктивності. Використання нативних компільованих мов програмування замість інтерпретованих середовищ чи важких гібридних фреймворків дозволяє досягти раціонального використання оперативної пам'яті та процесорного часу кінцевих станцій. Створення єдиної кодової бази для різних операційних систем на основі спеціалізованих інструментаріїв розробки гарантує ідентичний користувацький досвід та спрощує підтримку життєвого циклу програмного продукту. Нативні клієнтські додатки здатні безперебійно функціонувати навіть на застарілому корпоративному обладнанні без відчутних затримок інтерфейсу.

Забезпечення високого рівня конфіденційності вимагає глибокої інтеграції криптографічних підсистем безпосередньо у ядро розроблюваного додатка. Делегування функцій шифрування зовнішнім утилітам або мережевим екранам створює додаткові точки відмови та розширює площу потенційних атак. Надійна архітектура повинна включати реалізацію протоколів SSL/TLS на рівні самих сокетів для безперервного транспортного захисту, а також імплементацію

механізмів симетричного шифрування на рівні запитів до бази даних. Такий комплексний підхід гарантує, що жоден байт чутливої інформації не опиниться у відкритому вигляді ані в транзитному мережевому каналі, ані на фізичних накопичувачах сервера.

Проведений аналіз предметної області та існуючих технічних рішень обґрунтовує доцільність обраного напряму дослідження. Розробка кросплатформної клієнт-серверної системи на базі нативних технологій з використанням мережевих сокетів дозволяє усунути ключові недоліки поширених корпоративних платформ. Впровадження вбудованого дворівневого криптографічного захисту, що органічно поєднує безпеку передачі даних та безпеку їхнього зберігання, формує надійне ізольоване середовище для комунікації. Створення такого програмного комплексу є практично значущим кроком для розв’язання проблеми безпечного обміну даними у закритих корпоративних інфраструктурах.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1. Аналіз вимог до програмного комплексу та вибір методологічних підходів

Процес розробки захищеної архітектури комунікаційної системи вимагає детальної формалізації характеристик майбутнього програмного продукту. Згідно з міжнародними стандартами інженерії програмного забезпечення, зокрема документом ISO/IEC/IEEE 29148, етап визначення вимог є критичним кроком для узгодження технічних рішень із загальною метою проєкту [18]. Специфіка розбудови засобів ізольованого обміну даними обумовлює необхідність формування жорстких критеріїв до підсистем керування доступом, мережевої маршрутизації та обробки баз даних. Моделювання очікуваної поведінки програмного комплексу спирається на принципи інтегрованої безпеки, які ґрунтовно описані в академічних працях з проєктування розподілених систем [19].

Аналіз функціональних вимог визначає базові сценарії взаємодії користувачів із серверною частиною. Програмний комплекс повинен забезпечувати надійний механізм реєстрації та авторизації з підтримкою примусової зміни пароля при першому вході або після скидання облікових даних адміністратором. Реалізація цієї вимоги передбачає блокування доступу до основного функціоналу комунікатора до моменту успішного оновлення пароля користувачем. Базова логіка автентифікації доповнюється необхідністю ізоляції сесій, де кожне підключення жорстко прив’язується до унікального ідентифікатора та ідентифікатора поточної кімнати для унеможливлення перехресного перехоплення даних [20].

Основою комунікаційної підсистеми є забезпечення обміну текстовими повідомленнями у трьох типах віртуальних кімнат — публічних, приватних та кімнатах для прямого листування. Система повинна підтримувати механізми створення нових кімнат, запрошення учасників, а також розширений інструментарій роботи з повідомленнями, включаючи їх редагування через розсилку системних сповіщень, закріплення та завантаження історії з підтримкою пагінації. Обов’язковою функціональною вимогою є наявність вбудованої консолі адміністратора на стороні сервера, яка дозволяє в режимі реального часу керувати користувачами, змінювати їхні параметри та контролювати стан системи без необхідності прямого виконання запитів до системи управління базами даних [21].

Група нефункціональних вимог концентрується на аспектах безпеки, продуктивності та сумісності рішення, що розробляється. Абсолютним пріоритетом є забезпечення високого рівня конфіденційності, що вимагає імплементації протоколів транспортного шифрування на базі сертифікатів та інтеграції алгоритмів симетричного шифрування безпосередньо у логіку роботи з реляційними таблицями. Використання сучасного стандарту мови програмування C++ 23 та відповідного мультиплатформного фреймворку диктується вимогою щодо кросплатформеності клієнтського та серверного додатків, що має гарантувати їхню ідентичну роботу на різних операційних системах без необхідності переписування базового коду [22].

Окрема увага під час аналізу приділяється вимогам до стабільності мережевої взаємодії та обробки потокових даних. Програмний комплекс повинен бути стійким до проблеми фрагментації пакетів на рівні протоколу керування передачею, яка неминуче виникає під час пересилання об'ємних масивів інформації. Відповідність цим нефункціональним критеріям формує технічний фундамент для подальшого вибору конкретних архітектурних шаблонів та методологій програмної інженерії.

Методологічний базис розробки системи спирається на принципи ітеративного проєктування та компонентно-орієнтованої архітектури. Цей підхід дозволяє послідовно нарощувати функціонал програмного комплексу, починаючи від базової імплементації мережевого обміну і закінчуючи інтеграцією складних криптографічних протоколів. Формування архітектурного рішення базується на строгій типізації взаємодії між модулями, що гарантує високий рівень передбачуваності поведінки системи під час пікових навантажень. Проєктування логіки обробки запитів здійснювалося з урахуванням концепції слабкої зв'язності, яка дозволяє незалежно модифікувати клієнтські та серверні компоненти за умови збереження узгодженого формату обміну даними [23].

Аналіз специфіки комунікаційних процесів у закритих інфраструктурах обґрунтовує доцільність використання класичної клієнт-серверної архітектури на базі мережевих протоколів транспортного рівня. Вибір протоколу керування передачею гарантує надійну, впорядковану доставку потоку байтів між вузлами без втрат та дублювання інформації. Відмова від високорівневих протоколів прикладного рівня на користь прямих з'єднань через сокети мінімізує накладні витрати на обробку заголовків та усуває проблему надмірної ресурсоємності. Встановлення постійного двостороннього каналу зв'язку забезпечує мінімальні затримки під час трансляції повідомлень у реальному часі, що є критичним параметром для інтерактивних систем [24, 25].

Фундаментальним архітектурним рішенням є впровадження концепції централізованого управління даними, де серверний вузол виступає єдиною точкою довіри та зберігання стану системи. На відміну від децентралізованих одноранговых топологій, сервер повністю контролює процеси маршрутизації, перевіряє цілісність вхідних пакетів та авторизує кожну дію кінцевого користувача. Усі текстові повідомлення, конфігурації профілів та параметри віртуальних просторів зберігаються виключно у серверній базі даних. Це гарантує синхронізацію історії листування для всіх клієнтських додатків незалежно від

їхнього часу підключення до мережі та виключає можливість розсинхронізації станів через мережеві збої кінцевих станцій [26].

Використання сервера як єдиного центру прийняття рішень створює надійне підґрунтя для реалізації суворих механізмів управління обліковими записами та ізоляції просторів спілкування. Кожен активний мережевий сокет на стороні сервера жорстко асоціюється з унікальним ідентифікатором авторизованого користувача та ідентифікатором його поточної віртуальної кімнати. Завдяки цій прив'язці серверна логіка здатна безпомилково фільтрувати транзитний трафік, надсилаючи нові повідомлення та системні сповіщення виключно цільовим отримувачам. Такий підхід унеможливлює несанкціоноване перехоплення даних користувачами з інших кімнат і забезпечує суворий контроль над виконанням адміністративних команд у межах заданого сеансу зв’язку [25, 26].

2.2. Проєктування базової архітектури системи та мережевої взаємодії

Проєктування базової архітектури програмного комплексу базується на принципах модульності та чіткого розподілу відповідальності між системними компонентами. Для забезпечення масштабованості та спрощення подальшої підтримки кодової бази систему було логічно розділено на три ізольовані, але тісно взаємопов’язані частини: спільну бібліотеку Common, ядро обробки даних Server та користувацьку підсистему Client. Застосування такого архітектурного підходу дає змогу уникнути дублювання програмного коду, стандартизувати формати обміну інформацією та забезпечити незалежний розвиток серверної і клієнтської підсистем за умови суворого дотримання єдиних інтерфейсів взаємодії.

Модуль Common виступає фундаментальною статичною бібліотекою, яка об'єднує мережеві утиліти та базові абстракції для роботи обох частин програмного комплексу. У цій бібліотеці зосереджено реалізацію поведінкового патерну Команда для уніфікації обробки мережевих запитів.

Серверна частина спроєктована як багатопотокове ядро, що базується на обробці захищених мережевих з’єднань та відповідає за маршрутизацію інформаційних потоків усередині локальної мережі. Внутрішня архітектура сервера жорстко розділена на шар доступу до даних, який абстрагує виконання прямих запитів до системи управління базами даних через набір спеціалізованих репозиторіїв. Окремим елементом контролю є вбудована консоль адміністратора, що функціонує паралельно з мережевим ядром та дозволяє виконувати службові інструкції безпосередньо у серверному середовищі з найвищим рівнем привілеїв. Інтеграція сховища даних реалізована через центральний менеджер підключень, який забезпечує ізольоване виконання транзакцій та прозоре застосування алгоритмів шифрування до збережених повідомлень.

Підсистема Client розроблена з акцентом на мінімізацію споживання апаратних ресурсів кінцевих станцій та забезпечення гнучкої взаємодії користувача з термінальним інтерфейсом. Її архітектура базується на асинхронному читанні консольного вводу та безперервному прослуховуванні вхідного мережевого трафіку через об’єкт інкапсульованого захищеного сокета. Логіка парсингу команд реалізована через внутрішній реєстр, який перетворює текстові інструкції користувача на структуровані запити перед їхнім шифруванням та пересиланням транспортним каналом. На рівні клієнтського додатка свідомо виключена складна бізнес-логіка та локальне збереження історії листування, що перетворює його на безпечний термінал для трансляції дій та відображення актуального стану системи.

Концептуальна взаємодія між розробленими модулями відбувається за моделлю суворого обміну запитами та відповідями. Клієнтський додаток використовує утиліти спільної бібліотеки для формування інформаційного пакета і передає його зашифрованим транспортним каналом до сервера. Серверне ядро перехоплює потік байтів, дешифрує пакет, передає його до підсистеми команд для синтаксичного аналізу, після чого шар доступу до даних фіксує відповідні зміни у реляційних таблицях. Сформована позитивна відповідь або системне сповіщення

про зміну стану віртуальної кімнати знову інкапсулюється у формат спільної бібліотеки та розсилається визначеним активним клієнтам, успішно завершуючи цикл мережевої транзакції.

Проєктування мережевої взаємодії між компонентами програмного комплексу базується на використанні текстового формату обміну даними JSON [29]. Вибір цього стандарту зумовлений його легковажністю, незалежністю від апаратної архітектури та високою швидкістю серіалізації структурованої інформації. Кожна команда, яка генерується клієнтським додатком або надсилається сервером як системне сповіщення, інкапсулюється у стандартизований об'єкт. Цей об'єкт містить службові поля для ідентифікації типу запиту, параметри маршрутизації та безпосередньо корисне навантаження. Такий підхід дозволяє уніфікувати процес обробки мережевих пакетів, усуваючи необхідність створення складних бінарних протоколів прикладного рівня, та забезпечує гнучкість розширення системи новими типами команд без модифікації базового парсера.

Використання протоколу керування передачею TCP як транспортного фундаменту гарантує надійну доставку інформації, проте створює специфічну проблему обробки потокових даних [27, 28, 29]. Відповідно до специфікацій протоколу, TCP функціонує як безперервний потік байтів і не зберігає межі прикладних повідомлень під час маршрутизації мережею. Під час передачі значних обсягів інформації, зокрема масивів історії листування з великою кількістю записів, розмір сформованого пакета формату JSON часто перевищує максимальний розмір сегмента мережевого інтерфейсу. Це призводить до фрагментації вихідного повідомлення на рівні операційної системи, через що сервер або клієнт може отримати лише частину структурованого тексту за один цикл читання з мережевого сокета.

Спроба прямої десеріалізації фрагментованого пакета неминуче викликає критичну помилку синтаксичного аналізатора, оскільки обірваний рядок не є

валідним об’єктом. Для розв’язання проблеми втрати меж повідомлень було спроєктовано та імплементовано спеціалізований механізм буферизації на рівні прикладного програмного забезпечення. Архітектурне рішення передбачає виділення динамічного буфера для кожного активного мережевого з’єднання. Усі вхідні потоки байтів, що зчитуються з криптографічного сокета, спочатку акумулюються у відповідному локальному буфері без негайної передачі до підсистеми обробки команд.

Розроблений підхід гарантує абсолютну стійкість мережевої взаємодії до фрагментації трафіку та затримок маршрутизації. Поєднання безперервного потоку TCP з жорстким контролем цілісності прикладних повідомлень на стороні отримувача дозволяє безпечно передавати масиви даних будь-якого розміру через зашифровані канали SSL/TLS. Описана концепція буферизації формує надійний транспортний рівень, який виключає можливість втрати інформації або аварійного завершення роботи клієнтських додатків через некоректну обробку частково доставлених мережевих пакетів.

2.3. Проєктування логіки доступу до даних та підсистеми безпеки

Проєктування рівня доступу до даних базується на використанні реляційної системи управління базами даних PostgreSQL. Вибір цієї платформи зумовлений високим ступенем відповідності вимогам надійності, підтримкою складних типів даних та наявністю розширюваної архітектури для інтеграції криптографічних модулів. Шар доступу до даних спроєктовано як ізольований архітектурний рівень, що відокремлює бізнес-логіку серверної частини від специфічних структурованих запитів та особливостей реалізації сховища. Така декомпозиція забезпечує високу підтримуваність програмного коду та спрощує процес тестування окремих компонентів системи через чітке розмежування відповідальності.

Центральним елементом архітектури підсистеми збереження є патерн Репозиторій. Використання цього шаблону дало змогу інкапсулювати логіку взаємодії з реляційними таблицями всередині спеціалізованих класів, надаючи серверному ядру зручний інтерфейс для роботи з доменними об’єктами. Замість прямого маніпулювання мережевими з’єднаннями та виконання сирих текстових запитів у коді сервера, взаємодія відбувається через методи високого рівня абстракції. Це надає змогу повністю приховати деталі реалізації фізичної схеми даних та забезпечити централізоване впровадження криптографічних алгоритмів безпосередньо у методах збереження та вибірки інформації.

Логічна структура бази даних розділена на три незалежні сутності, кожна з яких обслуговується власним репозиторієм. UserRepository відповідає за управління обліковими записами користувачів, включаючи процеси реєстрації, автентифікації та контролю станів профілів. MessageRepository інкапсулює операції зі збереження текстових повідомлень, завантаження історії листування з підтримкою пагінації та управління метаданими. RoomRepository забезпечує логіку формування віртуальних кімнат різних типів, контроль прав доступу учасників та відстеження статусів прочитання повідомлень. Розділення монолітної логіки на ці сутності дозволяє уникнути надмірної складності класів та забезпечує гнучкість при розширенні функціоналу конкретної підсистеми без ризику порушення роботи інших модулів.

Координацію роботи всіх компонентів доступу до даних виконує компонент DatabaseManager. Цей елемент виступає центральним вузлом управління, що відповідає за ініціалізацію підключення до сервера PostgreSQL, налаштування параметрів сеансу та життєвий цикл об’єктів репозиторіїв. DatabaseManager забезпечує ін’єкцію необхідних конфігурацій, зокрема спільних ключів шифрування, у відповідні сутності під час їх створення. Завдяки використанню цього менеджера серверне ядро отримує єдину точку входу до підсистеми

збереження, що гарантує цілісність транзакцій та спрощує моніторинг стану бази даних у реальному часі через інтегровану систему логування.

Проєктування підсистеми безпеки мережевого трафіку базується на впровадженні протоколів SSL/TLS для створення захищеного каналу між клієнтськими додатками та сервером. Архітектурне рішення передбачає обов'язкове використання сертифікатів відкритого ключа на основі алгоритму RSA з довжиною ключа 2048 біт. Це забезпечує надійний захист від атак типу перехоплення трафіку та гарантує автентичність вузлів під час ініціалізації з'єднання. На рівні програмної реалізації захист інкапсулюється безпосередньо у класи мережевої взаємодії через використання бібліотеки OpenSSL — це уможливило прозоре шифрування всіх даних, які передаються у форматі JSON, без додаткового навантаження на логіку прикладного рівня.

Концепція захисту даних у стані спокою реалізується шляхом криптографічного перетворення чутливої інформації безпосередньо перед її фізичним збереженням на дискових накопичувачах сервера. Просте використання транспортного шифрування не вирішує проблему безпеки архіву листування у разі компрометації бази даних або отримання несанкціонованого доступу до файлової системи. Для нейтралізації цих ризиків спроєктовано механізм, який забезпечує збереження змісту повідомлень у зашифрованому вигляді всередині реляційних таблиць. Це гарантує конфіденційність інформації навіть за умови повного доступу сторонньої особи до вмісту накопичувачів або знімків пам'яті сервера баз даних.

Практична імплементація захисту змісту повідомлень базується на застосуванні алгоритмів симетричного шифрування безпосередньо на рівні виконання SQL-запитів. Для цього задіяна функціональність криптографічного розширення pgcrypto, яка дає змогу використовувати функції PGP_SYM_ENCRYPT для запису та PGP_SYM_DECRYPT для читання даних. Ключ шифрування передається до бази даних як динамічний параметр, що виключає його постійне зберігання у відкритому вигляді в системних журналах

СУБД. Використання типу даних BYTEA для поля тексту повідомлення дозволяє коректно фіксувати бінарні результати шифрування, забезпечуючи високу стійкість до спроб відновлення оригінального тексту без знання секретного ключа.

Захист облікових даних користувачів реалізується через незворотне криптографічне гешування паролів із застосуванням алгоритму SHA-256. Проєктне рішення виключає збереження паролів у відкритому вигляді — замість цього у базу даних записується результат обробки введеної послідовності за допомогою класу QCryptographicHash. Під час процедури автентифікації сервер порівнює результат гешування введеної комбінації з тими даними, що зберігаються у таблиці користувачів. Такий підхід унеможливлює відновлення оригінального пароля навіть адміністратором системи та захищає облікові записи від компрометації у разі витоку змісту реляційних таблиць.

Інтеграція описаних методів безпеки у єдину архітектуру формує ешелоновану систему захисту, де кожен рівень доповнює попередній. Транспортний рівень SSL/TLS забезпечує конфіденційність під час передачі, алгоритми гешування SHA-256 гарантують цілісність та безпеку автентифікації, а симетричне шифрування на базі pgcrypto захищає інформаційний актив у сховищі. Такий комплексний підхід до проєктування підсистеми безпеки дає змогу створити стійке до зовнішніх та внутрішніх загроз середовище для корпоративного обміну повідомленнями у локальних мережах.

2.4. Алгоритмічне забезпечення серверної частини та підсистеми адміністрування

Алгоритмічне забезпечення серверної частини базується на функціонуванні спеціалізованого об'єкта класу QSslServer, який забезпечує асинхронне прослуховування визначеного мережевого порту. Процес роботи ядра починається

з ініціалізації мережевого інтерфейсу та завантаження криптографічних сертифікатів і закритих ключів у конфігурацію безпеки. Після успішного запуску циклу очікування сервер переходить у режим фонової обробки подій операційної системи. При появі нового запиту від клієнта алгоритм автоматично створює екземпляр захищеного сокета, який успадковує всі налаштування безпеки сервера, та додає його до внутрішнього списку активних підключень для подальшого моніторингу трафіку.

Алгоритм обробки вхідного підключення передбачає обов’язковий етап встановлення зашифрованого каналу перед початком будь-якого прикладного обміну. Після фізичного підключення вузла ініціюється процедура криптографічного рукостискання, під час якої виконується верифікація сертифікатів та узгодження параметрів симетричного шифрування. У разі виявлення помилок у конфігурації SSL або невідповідності ключів алгоритм негайно розриває з'єднання та вилучає об'єкт сокета з пам'яті. Успішне завершення ініціалізації генерує сигнал готовності, після чого сокет переходить у режим очікування вхідних даних, а серверна логіка встановлює початкові метадані для нового вузла.

Логіка динамічного управління станом користувача базується на використанні механізму властивостей об’єктів для розширення стандартного інтерфейсу мережевого сокета. Після успішної перевірки облікових даних алгоритм виконує динамічну прив’язку унікального ідентифікатора користувача та ідентифікатора поточної віртуальної кімнати безпосередньо до об’єкта сокета за допомогою функцій запису властивостей. Це дозволяє серверному ядру миттєво отримувати контекст авторизації для кожного вхідного пакета без необхідності проведення повторних пошуків у базах даних або глобальних масивах станів. Такий підхід перетворює кожен сокет на автономний контейнер, що зберігає всі необхідні параметри сеансу протягом усього часу його активності.

Використання властивостей ідентифікатора користувача та ідентифікатора кімнати гарантує надійну логічну ізоляцію інформаційних потоків усередині сервера. Під час розсилки нових повідомлень або системних сповіщень алгоритм фільтрації виконує ітерацію по списку активних підключень, порівнюючи значення ідентифікатора кімнати цільового об’єкта зі значенням, закріпленим за кожним конкретним сокетом. Повідомлення передається лише тим вузлам, чиї параметри збігаються з контекстом події. Оскільки доступ до модифікації цих властивостей має виключно серверне ядро під час виконання валідних команд приєднання до кімнати, можливість несанкціонованого перехоплення трафіку між різними групами користувачів повністю виключається на рівні архітектури обробки подій.

Алгоритмічне забезпечення підсистеми адміністрування реалізується через незалежний модуль консолі, який функціонує в окремому потоці виконання паралельно з основним мережевим ядром. Алгоритм роботи адміністративного термінала базується на циклічному читанні стандартного потоку вводу сервера. Кожен введений рядок передається до спеціалізованого лексичного аналізатора, який розбиває команду на базові лексеми з урахуванням екранованих символів та пробілів у лапках. Отримана послідовність лексем маршрутизується до відповідного обробника через патерн Команда.

Алгоритми обробки команд адміністрування облікових записів інкапсулюють складну логіку взаємодії з репозиторіями даних. Під час виконання команди додавання нового користувача алгоритм спочатку перевіряє унікальність логіна, генерує криптографічний геш для заданого пароля та створює відповідний запис у базі даних. Алгоритм видалення користувача не виконує фізичного знищення записів з таблиць для збереження цілісності історії листування, натомість він встановлює позначку логічного видалення та ініціює примусове розірвання активних мережевих сесій для вказаного облікового запису. Зміна параметрів профілю, зокрема пароля або публічного імені, виконується через транзакційне оновлення відповідних полів у базі з обов’язковою активацією

прапорця примусової зміни пароля під час наступної авторизації, якщо операція була ініційована адміністратором.

Окремим критичним компонентом серверної логіки є алгоритм синхронізації станів у публічних кімнатах, зокрема у глобальному просторі спілкування. Оскільки публічні кімнати є відкритими для всіх авторизованих користувачів, виникає проблема коректного підрахунку непрочитаних повідомлень для тих вузлів, які ще не виконували явної дії приєднання. Для розв'язання цієї проблеми розроблено алгоритм автоматичної реєстрації, який спрацьовує під час першого успішного входу користувача до системи.

Якщо запис про участь відсутній, алгоритм автоматично генерує та виконує транзакцію приєднання, встановлюючи початковий маркер останнього прочитаного повідомлення на поточний кінець історії листування. Ця дія перетворює глобальну кімнату з абстрактного публічного простору на конкретний об’єкт моніторингу для профілю користувача. Завдяки цій автоматичній синхронізації алгоритм підрахунку непрочитаних повідомлень отримує коректну точку відліку для кожного клієнта. Під час запиту на отримання списку кімнат сервер виконує порівняння ідентифікатора останнього повідомлення у кімнаті з маркером прочитання, що зберігається у таблиці учасників, гарантуючи точне відображення статусу нових повідомлень в інтерфейсі клієнтського додатка без необхідності розсилання додаткових мережевих пакетів.

2.5. Алгоритми клієнтської взаємодії, парсингу команд та обробки криптографічних протоколів

Алгоритмічне забезпечення підсистеми клієнтської взаємодії базується на використанні централізованого реєстру команд, що забезпечує гнучку обробку користувацького вводу через термінальний інтерфейс. В основі архітектури лежить принцип делегування, де кожна текстова інструкція, введена користувачем,

проходить через багаторівневий фільтр розпізнавання та валідації. Такий підхід дозволяє відокремити інтерфейсну логіку від безпосередньої реалізації мережевих запитів — це гарантує високу стабільність роботи клієнтського додатка та спрощує розширення функціональних можливостей системи новими типами команд без модифікації ядра обробки подій.

Реалізація поведінкового патерну Команда дозволяє формалізувати кожну дію користувача як окрему програмну сутність із власним набором параметрів та логікою виконання. Реєстр команд будується за ієрархічним принципом, де кореневий об’єкт містить посилання на підпорядковані модулі, кожен з яких відповідає за конкретну функцію — від автентифікації до управління віртуальними кімнатами. Під час отримання вводу алгоритм виконує пошук відповідного об’єкта в реєстрі, що забезпечує швидке розпізнавання інструкцій та автоматичну генерацію довідкової інформації про синтаксис кожної команди. Застосування цієї абстракції дозволяє інкапсулювати деталі формування мережевих пакетів усередині окремих класів, мінімізуючи ризики виникнення побічних ефектів при зміні протоколу обміну.

Алгоритм парсингу складних команд забезпечує коректну обробку вхідних рядків з урахуванням специфічних вимог до передачі текстових аргументів у консольному середовищі. Особлива увага приділяється логіці розділення рядка на окремі лексеми, де пробіли розглядаються як роздільники параметрів, за винятком випадків їхнього перебування всередині подвійних лапок. Процес розбору базується на послідовному лексичному аналізі символів, що дозволяє вилучати складні описи кімнат або зміст довгих повідомлень як єдині цілісні аргументи. Такий алгоритмічний підхід дозволяє уникнути некоректної інтерпретації команд, де назва або параметри містять службові символи, гарантуючи точну передачу інформації від клієнта до сервера без спотворення змісту.

Після успішного синтаксичного розбору рядка алгоритм переходить до етапу семантичної перевірки аргументів, яка виконується індивідуально для кожного

типу команди. Процес включає конвертацію текстових послідовностей у необхідні типи даних, зокрема числові ідентифікатори або булеві прапорці. У разі виявлення некоректних параметрів, таких як невалідний ідентифікатор повідомлення або непідтримуваний тип кімнати, алгоритм перериває виконання команди та ініціює вивід діагностичного сповіщення в термінал користувача. Це запобігає відправці завідомо помилкових запитів у мережу, знижуючи навантаження на серверну частину системи.

Завершальна фаза клієнтської взаємодії передбачає трансформацію перевірених даних у структурований інформаційний пакет перед його відправкою в мережевий канал. Ця операція виконується асинхронно, що дозволяє клієнтському додатку залишатися чуйним до подальшого вводу користувача навіть під час очікування відповіді від сервера. Описана послідовність дій формує надійний алгоритмічний ланцюжок, який забезпечує точне перетворення намірів користувача у валідні мережеві транзакції в умовах зашифрованого з’єднання.

Алгоритмічне забезпечення користувацького інтерфейсу в термінальному середовищі стикається з фундаментальними обмеженнями потокового виводу даних. Традиційні методи динамічного оновлення контенту, властиві графічним інтерфейсам, є технічно складними для реалізації у стандартній консолі через неможливість довільного доступу до раніше виведених рядків без використання спеціалізованих низькорівневих бібліотек керування екраном. Для розв’язання цієї проблеми було розроблено алгоритм асинхронних системних сповіщень, який замінює пряме редагування тексту в уже виведеному блоці на трансляцію службових повідомлень про зміну стану інформаційного об’єкта. Такий підхід дозволяє зберігати лінійність логування подій, забезпечуючи при цьому актуальність відображуваних даних для користувача через послідовне додавання нових статусів до потоку виводу.

Алгоритм обробки сигналів про редагування або зміну статусу повідомлення іншими учасниками базується на безперервному моніторингу вхідного мережевого

потоку у форматі JSON. Під час отримання пакета з типом команди про редагування клієнтський додаток ініціює процедуру розбору ідентифікатора повідомлення та нового змісту тексту. Замість спроби модифікації старого рядка алгоритм виконує вивід службового сповіщення, яке містить унікальний номер повідомлення та його оновлену версію. Аналогічна логіка застосовується під час обробки сигналів закріплення або відкріплення повідомлень. Отримання сигналу про зміну статусу пріоритетності викликає спрацювання обробника, який інформує користувача про факт фіксації конкретного повідомлення в історії кімнати — це дає змогу підтримувати спільний контекст спілкування без порушення структури консольного виводу та без необхідності повного перемальовування вікна термінала.

Процес ініціалізації транспортного шифрування на стороні клієнта є критичним етапом, що передує будь-якій передачі прикладних даних. Алгоритм починається з конфігурації об’єкта захищеного сокета, де встановлюються параметри перевірки сертифікатів та вибору дозволених криптографічних протоколів. Клієнтський додаток використовує метод встановлення зашифрованого з’єднання, який ініціює низькорівневу процедуру рукостискання з сервером. Під час цього процесу клієнт отримує відкритий ключ сервера, виконує його валідацію та бере участь у генерації спільних сесійних ключів для подальшого симетричного шифрування трафіку. Використання асинхронної моделі дозволяє додатку продовжувати роботу в очікуванні підтвердження безпеки каналу, не блокуючи основний потік виконання та обробку вводу користувача.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Обґрунтування вибору технологічного стеку та інструментальних засобів розробки

Для практичної реалізації програмного комплексу обрано мову програмування C++ стандарту 2023 року. Такий вибір зумовлений необхідністю забезпечення високої продуктивності під час виконання інтенсивних обчислювальних операцій, зокрема криптографічних перетворень та обробки мережевих пакетів у реальному часі. Використання стандарту C++23 дає змогу задіяти сучасні мовні конструкції для підвищення безпеки коду, покращення читабельності та оптимізації роботи з пам’яттю, що критично важливо для серверних застосунків з тривалим циклом роботи. Мова C++ надає контроль над системними ресурсами, що дозволяє мінімізувати накладні витрати порівняно з мовами, які використовують віртуальні машини або інтерпретатори [22].

Технологічним фундаментом розробки виступає фреймворк Qt версії 6.0 та вище. Вибір даного інструментарію обґрунтований його кросплатформеною природою, що забезпечує можливість компіляції та стабільного функціонування коду в різних операційних системах без внесення суттєвих змін до архітектури. Об’єктно-орієнтована модель Qt у поєднанні з механізмом сигналів та слотів дозволяє ефективно реалізувати асинхронну обробку подій, що є базовою вимогою для побудови відгукливих клієнт-серверних систем. Використання нативних засобів фреймворку гарантує високу швидкість виконання операцій введення-виведення та надійну інтеграцію з системними API [30].

Забезпечення мережевої взаємодії реалізовано за допомогою модуля Qt Network. Даний компонент надає високорівневі абстракції для роботи з TCP-сокетами, що значно спрощує процес керування мережевими з’єднаннями та

передачі потокових даних. Особлива увага приділена класу QSslSocket, який інтегрує підтримку протоколів SSL/TLS для шифрування трафіку. Практична необхідність використання цього модуля полягає у можливості створення стійких до фрагментації каналів зв'язку та прозорого керування сертифікатами безпеки без необхідності глибокої ручної взаємодії з низькорівневими системними бібліотеками [31].

Робота з даними та обмін інформацією базуються на модулях Qt Sql та Qt Core. Модуль Qt Sql забезпечує уніфікований інтерфейс для взаємодії з реляційними системами управління базами даних, що дозволяє ефективно виконувати структуровані запити до PostgreSQL та обробляти результати вибірок у зручному об’єктному форматі. Використання засобів Qt Core для парсингу даних у форматі JSON є критичним для забезпечення швидкої серіалізації та десеріалізації повідомлень. Таке поєднання інструментів дає змогу побудувати гнучку систему обробки команд, де кожен мережевий пакет проходить валідацію та синтаксичний аналіз з мінімальними затримками [30, 32].

Порівняльний аналіз демонструє переваги обраного стеку над альтернативними рішеннями на базі мов Python, Java або Node.js. Використання компільованої мови C++ усуває ризики, пов'язані з непередбачуваними затримками через роботу збирача сміття, що характерно для керованих середовищ. Можливість створення статично скомпільованого бінарного файлу полегшує розгортання системи в ізольованих локальних мережах, оскільки не потребує встановлення додаткових інтерпретаторів або віртуальних машин. Високий рівень контролю над криптографічними операціями та пряма робота з пам’яттю роблять комбінацію C++ та Qt оптимальним вибором для побудови систем, де вимоги до безпеки та стабільності є пріоритетними [22, 33].

Вибір PostgreSQL як основної системи керування базами даних обумовлений її високою надійністю, повною відповідністю принципам ACID та здатністю ефективно обробляти складні структури даних. Для реалізації криптографічного

захисту змісту повідомлень на рівні сховища використано спеціалізоване розширення pgcrypto. Це рішення дозволяє виконувати симетричне шифрування та дешифрування інформації безпосередньо у SQL-запитах, що суттєво підвищує загальний рівень безпеки системи. Застосування pgcrypto забезпечує надійний захист даних у стані спокою, унеможливлюючи отримання доступу до приватного листування навіть у разі прямого витоку файлів бази даних або компрометації фізичного носія інформації.

Забезпечення конфіденційності та цілісності мережевого трафіку реалізовано на основі бібліотеки OpenSSL, яка є світовим стандартом у сфері криптографічного захисту інформації. Вибір цього інструментарію зумовлений його глибокою інтеграцією з мережевими модулями Qt та підтримкою найсучасніших протоколів шифрування. OpenSSL надає необхідний набір засобів для генерації ключів, роботи з сертифікатами та імплементації захищених сокетних з'єднань. Використання даної бібліотеки гарантує стійкість системи до атак типу перехоплення даних та забезпечує надійну автентифікацію вузлів під час встановлення сеансу зв'язку в локальній мережі.

Для організації ефективного моніторингу роботи серверної частини та відстеження критичних подій у реальному часі обрано бібліотеку spdlog. Дане рішення вирізняється надзвичайно високою швидкістю виконання операцій логування, що дозволяє фіксувати стан системи без відчутного впливу на продуктивність мережевого ядра. Практична роль spdlog полягає у забезпеченні детальної звітності про процеси авторизації, мережеві помилки та дії адміністратора в консолі. Гнучкість налаштування рівнів логування та підтримка різних форматів виводу роблять цю бібліотеку оптимальним засобом для діагностики та підтримки стабільності серверного додатка.

Керування процесом компіляції та автоматизація збірки кросплатформеного проєкту здійснюється за допомогою системи CMake. Вибір цього інструментарію обґрунтований його здатністю ефективно керувати зовнішніми залежностями,

автоматично налаштовувати шляхи до бібліотек Qt, OpenSSL та PostgreSQL у різних операційних системах. CMake дає змогу розділити процес опису архітектури проєкту від специфіки конкретного середовища розробки, що забезпечує легке перенесення коду між платформами. Використання стандартизованих скриптів збірки гарантує цілісність усіх компонентів програмного комплексу та спрощує інтеграцію нових модулів на різних етапах розробки.

3.2 Практична імплементація серверної, клієнтської частин та підсистеми криптографії

Практична імплементація серверної частини базується на функціонуванні централізованого компонента DatabaseManager, який виконує роль головного вузла керування підключенням до реляційної бази даних PostgreSQL. Під час ініціалізації сервера цей модуль забезпечує встановлення з’єднання, налаштування драйверів та імплементацію схеми даних через виконання стартових запитів для створення таблиць і розширень. Структура рівня доступу до даних реалізована через відокремлені репозиторії, що дає змогу інкапсулювати специфічну логіку всередині спеціалізованих об'єктів. Кожен репозиторій отримує доступ до активного підключення та забезпечує виконання транзакцій для роботи з профілями користувачів, історією повідомлень або параметрами віртуальних кімнат.

Керування станом автентифікації на сервері реалізовано через багаторівневу перевірку прав доступу, де ключовим елементом є механізм примусової зміни пароля. У структурі таблиці користувачів передбачено логічний прапорець, який вказує на стан актуальності автентифікаційних даних. Під час обробки запиту на авторизацію серверне ядро аналізує значення цього прапорця безпосередньо після верифікації криптографічного геша. Якщо прапорець має від’ємне значення, сервер обмежує контекст сесії, не встановлюючи для мережевого сокета статус повної

авторизації. У такому стані обробник вхідних пакетів відхиляє будь-які команди, окрім запиту на зміну пароля, що гарантує неможливість доступу до приватного листування або списків кімнат до моменту успішного оновлення облікових даних.

Логіка блокування функцій чату інтегрована безпосередньо в диспетчер вхідних команд на стороні сервера. Перед викликом будь-якого методу обробки повідомлень або запиту історії система виконує перевірку динамічних властивостей об’єкта сокета. Доступ до функціональних алгоритмів надається виключно за умови наявності підтвердженого статусу успішної авторизації та відсутності активної вимоги щодо зміни пароля. Така архітектурна реалізація дозволяє забезпечити сувору ізоляцію функцій системи та унеможливити виконання несанкціонованих дій через маніпуляції мережевими пакетами на етапі ініціалізації сеансу.

Система логування подій на сервері побудована на базі бібліотеки spdlog, інтегрованої у всі ключові модулі програмного комплексу. Практична імплементація передбачає фіксацію кожного етапу життєвого циклу сервера — від ініціалізації мережевого порту до завершення клієнтських сесій. У межах роботи з базою даних забезпечено реєстрацію критичних помилок виконання запитів та успішного встановлення з’єднання. На рівні мережевого ядра реалізовано детальний моніторинг вхідних підключень, автентифікаційних спроб та помилок транспортного рівня. Використання асинхронних логерів дозволяє вести безперервний запис системного стану без впливу на швидкість обробки мережевих повідомлень, що гарантує стабільність роботи системи під високим навантаженням.

Практична імплементація підсистеми адміністрування реалізована через відокремлений модуль консолі, який функціонує незалежно від основного циклу обробки мережевих подій. Обробка адміністративних інструкцій виконується шляхом асинхронного читання стандартного потоку вводу сервера з подальшою передачею отриманих текстових рядків до лексичного аналізатора. Зіставлення

розпізнаних команд із відповідними методами керування базою даних дозволяє на практиці здійснювати реєстрацію нових користувачів, зміну їхніх облікових даних та логічне видалення профілів без зупинки роботи серверного ядра. Такий підхід забезпечує безперервність надання комунікаційних послуг під час виконання критичних операцій з обслуговування інфраструктури.

Клієнтський застосунок імплементовано на базі асинхронної моделі взаємодії, де читання користувацького вводу та моніторинг мережевого трафіку відбуваються паралельно через механізми системних сповіщень. Центральним елементом обробки клієнтських інструкцій виступає реєстр команд, побудований на основі поліморфної ієрархії класів. Кожна підтримувана операція інкапсульована в окремий об’єкт, який містить правила валідації аргументів та логіку формування вихідного пакета у форматі JSON. Розподіл відповідальності між мережевим класом клієнта та обробниками конкретних команд гарантує високу стабільність роботи термінального інтерфейсу та виключає можливість відправки некоректно сформованих запитів до сервера.

Впровадження транспортного шифрування виконано за допомогою спеціалізованих класів захищених мережевих сокетів. На стороні сервера реалізовано алгоритм читання локальних файлів відкритого сертифіката та закритого ключа стандарту RSA, які після успішної валідації завантажуються у глобальний конфігураційний об’єкт безпеки. Клієнтська імплементація використовує метод встановлення зашифрованого підключення, який ініціює процедуру криптографічного рукостискання відразу після фізичного з’єднання з мережевим портом. Для забезпечення сумісності в умовах ізольованих мереж на клієнті імплементовано обробник помилок верифікації, який дозволяє безпечно ігнорувати попередження, пов’язані з використанням самопідписаних сертифікатів, не перериваючи при цьому процес обміну сесійними ключами.

Практична реалізація захисту автентифікаційних даних базується на безпосередньому використанні криптографічної функції SHA-256 під час усіх

операцій з паролями. Під час спроби авторизації або реєстрації нового профілю введений пароль у вигляді текстового рядка обов’язково конвертується у масив байтів формату UTF-8. Далі цей масив передається до статичного методу класу QCryptographicHash, який виконує незворотне математичне перетворення і повертає результуючий дайджест у вигляді шістнадцяткового рядка. Саме це зашифроване значення передається до репозиторію бази даних для порівняння з існуючим записом або для безпечного збереження, що гарантує повну відсутність паролів у відкритому вигляді на будь-якому етапі роботи серверної частини.

3.3 Тестування програмного продукту та оцінка результатів роботи

Методологія тестування розробленого програмного комплексу базувалася на комплексному підході, який охоплював модульне тестування ізольованих компонентів та інтеграційне тестування системи в умовах, максимально наближених до реального експлуатаційного середовища локальної мережі. Для перевірки стабільності архітектурних рішень було розгорнуто тестовий стенд, що складався з виділеного сервера бази даних PostgreSQL та кількох клієнтських станцій. Основною метою етапу тестування стала верифікація коректності роботи механізмів криптографічного захисту, перевірка пропускної здатності мережевого ядра та оцінка стійкості системи управління станами до нетипових сценаріїв використання [22, 30].

Окрему увагу під час інтеграційного тестування було приділено перевірці надійності мережевого рівня при трансляції масивів даних великого обсягу. Критичним сценарієм стала операція завантаження історії повідомлень для віртуальних кімнат з тривалим періодом активності. Тестування проводилося шляхом штучної генерації кількох тисяч текстових записів у репозиторії бази даних з подальшим ініціюванням запитів на отримання історії від різних клієнтських додатків. Цей процес дав змогу оцінити ефективність алгоритмів пагінації та

визначити граничні обсяги пам’яті, які споживаються об’єктами десеріалізації на кінцевих вузлах під час пікових навантажень [31].

У ході проведення навантажувального тестування трансляції великих історій листування було виявлено критичну проблему маршрутизації на рівні транспортного протоколу TCP. Під час пересилання значних обсягів інформації, що перевищували стандартний розмір сегмента мережевого вікна, операційна система автоматично виконувала фрагментацію єдиного повідомлення формату JSON на кілька дрібних пакетів. Клієнтський додаток, намагаючись десеріалізувати обірваний текстовий рядок відразу після отримання першого фрагмента, фіксував фатальну помилку синтаксичного аналізатора, що призводило до втрати даних [32].

Для вирішення виявленої проблеми маршрутизації було впроваджено та протестовано механізм локальної буферизації мережевого потоку. Результати повторних випробувань підтвердили високу ефективність розробленого алгоритму перевірки цілісності структури JSON перед її передачею до парсера. Акумуляція фрагментованих пакетів у виділеному буфері сокета з наступним підрахунком фігурних дужок дозволила системі гарантовано формувати коректні об’єкти незалежно від інтенсивності фрагментації на рівні комутаційного обладнання. Успішне проходження серії стрес-тестів довело здатність оновленого мережевого ядра стабільно обробляти потоки даних будь-якого розміру без порушення цілісності прикладного протоколу [32, 33].

Результати функціонального тестування підсистеми AdminConsole підтвердили повну відповідність розробленого інструментарію вимогам щодо оперативного управління серверною частиною. Під час випробувань було перевірено швидкість та точність виконання команд реєстрації нових користувачів, зміни паролів та логічного видалення профілів безпосередньо через консольний інтерфейс сервера у реальному часі. Тестування продемонструвало, що зміни у реляційній базі даних відображаються миттєво, а активні мережеві сесії коректно реагують на адміністративні дії, зокрема ініціюють розірвання з’єднання у разі

блокування облікового запису. Висока стабільність лексичного аналізатора дозволила успішно обробляти складні параметри команд із використанням пробілів та спеціальних символів без збоїв у роботі ядра.

Аналіз логіки синхронізації в публічних кімнатах підтвердив коректність роботи алгоритму автоматичної реєстрації користувачів у таблиці room-members під час першої успішної автентифікації. Було верифіковано, що система безпомилково встановлює початковий маркер останнього прочитаного повідомлення, що дозволяє уникнути некоректного відображення лічильника нових повідомлень при ініціалізації клієнтського інтерфейсу. Перевірка механізму підрахунку unread-count показала повну відповідність між кількістю фактично надісланих у кімнату повідомлень та даними, що відображаються у списку доступних просторів після тривалої відсутності користувача в мережі. Це підтверджує стабільність роботи тригерів та запитів на рівні шару доступу до даних.

Оцінка технічної реалізації системи захисту підтвердила ефективність ешелонованого підходу до гарантування конфіденційності. Під час стрес-тестування транспортного рівня було підтверджено стійкість SSL/TLS з’єднань до спроб перехоплення трафіку інструментами мережевого аналізу. Перевірка підсистеми захисту даних у стані спокою продемонструвала, що зміст повідомлень у реляційних таблицях залишається повністю зашифрованим і недоступним для читання без використання валідного ключа шифрування, переданого сервером. Результати тестування хешування паролів за алгоритмом SHA-256 підтвердили неможливість відновлення оригінальних автентифікаційних даних навіть за умови отримання повного доступу до файлів бази даних.

Загальна оцінка розробленого продукту дозволяє стверджувати, що створена клієнт-серверна система є стабільним та надійно захищеним рішенням, яке повністю відповідає поставленим технічним вимогам. Програмний комплекс демонструє високу продуктивність під час обробки потокових даних та стійкість

до фрагментації пакетів у локальних мережах. Поєднання нативної реалізації на базі сокетів та вбудованих засобів криптографічного захисту робить систему повністю готовою до розгортання в ізольованих корпоративних інфраструктурах. Отримані результати підтверджують можливість використання даного продукту як надійного середовища для конфіденційного обміну повідомленнями без залучення сторонніх хмарних сервісів.

ВИСНОВКИ

Мета кваліфікаційної роботи, яка полягала у створенні надійного програмного комплексу для захищеного обміну повідомленнями в межах локальних мереж, була повністю досягнута, а всі поставлені дослідницькі та практичні завдання успішно виконані. Результатом роботи стала не просто програмна реалізація базового чату, а комплексна масштабована система, здатна забезпечити високий рівень конфіденційності без необхідності підключення до глобальної мережі. Це має критичне значення для корпоративних інфраструктур та ізольованих середовищ, де витік інформації через зовнішні сервери є неприпустимим ризиком. Створений продукт розв’язує фундаментальну проблему безпечної комунікації, надаючи користувачам повний контроль над власною інфраструктурою та даними.

Теоретичний аналіз предметної області переконливо довів необхідність відмови від використання публічних хмарних сервісів на користь автономних локальних рішень у випадках, коли висуваються жорсткі вимоги до безпеки. Дослідження підтвердило, що делегування обробки корпоративного або приватного листування стороннім провайдерам завжди супроводжується ризиком компрометації через вразливості архітектури постачальника послуг. Розробка ізольованої системи дозволила повністю нівелювати загрози, пов’язані з перехопленням трафіку на магістральних каналах зв’язку та несанкціонованим доступом до серверів баз даних ззовні.

Оцінка архітектурних результатів підтверджує правильність вибору класичної клієнт-серверної моделі взаємодії на базі протоколів транспортного рівня. На відміну від децентралізованих топологій, наявність єдиного довіреного серверного вузла забезпечила суворий контроль за процесами маршрутизації, перевіркою цілісності пакетів та автентифікацією учасників. Спроєктована модель дала змогу реалізувати концепцію єдиної точки управління, де сервер повністю контролює матрицю прав доступу, логічно ізолює віртуальні простори спілкування

та синхронізує історію повідомлень для всіх підключених терміналів. Це створило надійний фундамент для запобігання внутрішнім атакам та унеможливило несанкціоноване перехоплення інформаційних потоків між авторизованими вузлами в мережі.

Вибір технологічного стеку, що включає мову програмування C++ стандарту 2023 року, кросплатформений фреймворк Qt 6 та систему управління базами даних PostgreSQL, виявився оптимальним для реалізації поставлених інженерних завдань. Використання компільованої мови у поєднанні з нативними інструментами мережевої обробки забезпечило високу продуктивність управління асинхронними з'єднаннями та мінімізувало затримки під час криптографічних перетворень у реальному часі. Інтеграція реляційної бази даних дала змогу перенести частину логіки безпеки безпосередньо на рівень сховища, гарантуючи стійкий захист збереженої історії листування. Створена комбінація програмних технологій сформувала гнучку, ресурсоефективну та масштабовану основу для розгортання захищеного комунікаційного середовища на різних операційних системах без втрати стабільності роботи.

Практична імплементація та результати всебічного тестування підтвердили повну працездатність усіх модулів програмного комплексу та високу ефективність обраних алгоритмічних рішень. Випробування в умовах реальної локальної мережі продемонстрували стійкість мережевого ядра до інтенсивних навантажень та коректність обробки потокових даних. Особливо важливим результатом стало успішне розв'язання проблеми фрагментації пакетів на транспортному рівні за допомогою розробленого механізму динамічної буферизації. Це дало змогу забезпечити безперебійну передачу об'ємних масивів історії повідомлень без ризику виникнення фатальних помилок синтаксичного аналізу, що підтверджує стабільність архітектури мережевої взаємодії.

Значення отриманих результатів полягає у створенні багаторівневої системи захисту, де поєднання транспортного шифрування та концепції захисту даних у

стані спокою сформувало надійний бар’єр проти зовнішніх та внутрішніх загроз. Впровадження протоколів SSL/TLS із використанням 2048 — бітних сертифікатів гарантувало повну конфіденційність трафіку під час передачі, а інтеграція симетричного шифрування безпосередньо в реляційні таблиці бази даних забезпечила захист інформаційного активу навіть у разі фізичного витоку файлів сховища. Доведено, що використання криптографічних засобів на різних рівнях архітектури не призвело до критичного зниження продуктивності системи, що робить розроблене рішення оптимальним для використання в умовах суворих вимог до безпеки корпоративної інформації.

Практичне значення розробленого продукту визначається його повною готовністю до розгортання в ізольованих інфраструктурах без необхідності залучення сторонніх сервісів або складного додаткового налаштування. Наявність розвиненої підсистеми адміністрування у реальному часі дозволяє оперативно керувати обліковими записами та правами доступу, забезпечуючи гнучкість обслуговування системи без переривання комунікаційних процесів. Кросплатформена природа клієнтського та серверного додатків гарантує ідентичність їхнього функціонування у гетерогенних мережевих середовищах, що робить програмний комплекс універсальним інструментом для побудови внутрішніх захищених каналів зв’язку в організаціях будь-якого масштабу.

Перспективи подальших досліджень та розвитку проєкту пов’язані з розширенням функціональних можливостей та посиленням криптографічної стійкості системи. Пріоритетним напрямком є впровадження протоколів наскрізного шифрування для забезпечення абсолютної приватності, де доступ до змісту листування матимуть виключно кінцеві учасники діалогу без технічної можливості розшифрування даних на стороні сервера. Подальша розробка передбачає створення мобільних клієнтів для операційних систем Android та iOS, а також додавання підтримки потокової передачі медіаданих та голосового зв’язку за допомогою протоколів реального часу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://researcheurope.org/book-129/
2. Stallings W. Cryptography and Network Security: Principles and Practice. 8th ed. Pearson, 2023. 833 p.
3. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. 20th Anniversary Edition. Wiley, 2015. 784 p.
4. Горбенко І. Д., Горбенко Ю. І.. «КРИПТОЛОГІЯ. ТЕОРІЯ. ПРАКТИКА. ЗАСТОСУВАННЯ». - 2012.
5. Ross R. S. Managing Information Security Risk: Organization, Mission, and Information System View. NIST Special Publication 800-39, 2011. 88 p.
6. Katz J., Lindell Y. Introduction to Modern Cryptography. 3rd ed. CRC Press, 2020. 648 p.
7. Anderson R. Security Engineering: A Guide to Building Dependable Distributed Systems. 3rd ed. Wiley, 2020. 1232 p.
8. Ferguson N., Schneier B., Kohno T. Cryptography Engineering: Design Principles and Practical Applications. Wiley, 2010. 384 p.
9. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. IETF, 2018. 160 p.
10. Boneh D., Shoup V. A Graduate Course in Applied Cryptography. Version 0.6. Stanford University, 2023. 1130 p.
11. Rivest R. L., Shamir A., Adleman L. A method for obtaining digital signatures and public—key cryptosystems. Communications of the ACM, 1978. Vol. 21, No. 2. P. 120-126.
12. National Institute of Standards and Technology. Secure Hash Standard (SHS). FIPS PUB 180-4. U.S. Department of Commerce, 2015. 36 p.
13. Krawczyk H., Bellare M., Canetti R. HMAC: Keyed-Hashing for Message Authentication. RFC 2104. IETF, 1997. 11 p.

14. Bishop M. Computer Security: Art and Science. 2nd ed. Addison-Wesley Professional, 2018. 1440 p.
15. Barker E. Recommendation for Key Management. NIST Special Publication 800-57 Part 1 Revision 5. U.S. Department of Commerce, 2020. 152 p.
16. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. McGraw-Hill Education, 2019. 1376 p.
17. Stallings W., Brown L. Computer Security: Principles and Practice. 4th ed. Pearson, 2017. 986 p.
18. ISO/IEC/IEEE 29148:2018. Systems and software engineering — Life cycle processes — Requirements engineering. International Organization for Standardization, 2018. 104 p.
19. Sommerville I. Software Engineering. 10th ed. Pearson, 2015. 816 p.
20. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. McGraw-Hill Education, 2019. 629-638 p.
21. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Addison-Wesley Professional, 2021. 497 p.
22. Stroustrup B. A Tour of C++. 3rd ed. Addison-Wesley Professional, 2022. 320 p.
23. Richards M. Software Architecture Patterns. O'Reilly Media, 2015. 58 p.
24. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 8th ed. Pearson, 2021. 775 p.
25. Stevens W. R., Fenner B., Rudoff A. M. UNIX Network Programming, Volume 1: The Sockets Networking API. 3rd ed. Addison—Wesley Professional, 2003. 1021 p.
26. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017. 616 p.
27. Postel J. Transmission Control Protocol. RFC 793. IETF, 1981. 91 p.
28. Fall K. R., Stevens W. R. TCP/IP Illustrated, Volume 1: The Protocols. 2nd ed. Addison-Wesley Professional, 2011. 1056 p.

29. Bray T. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259. IETF, 2017. 16 p.
30. The Qt Company. Qt Documentation. URL: https://doc.qt.io/ (дата звернення: 10.05.2024).
31. Qt Network Module Documentation. The Qt Company, 2024.
32. Qt Core Module Documentation. The Qt Company, 2024.
33. Williams A. C++ Concurrency in Action. 2nd ed. Manning Publications, 2019. 616 p.
34. https://mintech-conf.lntu.edu.ua/en/home/