

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій

(повне найменування інституту, назва факультету)

Кафедра інформаційної безпеки та наноелектроніки

(повна назва кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему Розробка захищеного P2P чату з шифруванням та автентифікацією
повідомлень

Development of secure P2P chat with message encryption and authentication

Спеціальності 125 Кібербезпека

(код і назва напрямку підготовки, спеціальності)

Освітня програма (спеціалізація)

Системи технічного захисту інформації,
автоматизація її обробки

БАЙША В.К

(ПРИЗВИЩЕ та ініціали)

Керівник НЕЛАСА Г.В

(ПРИЗВИЩЕ та ініціали)

Рецензент МАЛИЙ О.Ю.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій
Кафедра інформаційної безпеки та наноелектроніки
Ступінь вищої освіти бакалавр
Спеціальність 125 Кібербезпека
(код і найменування)
Освітня програма (спеціалізація) Системи технічного захисту інформації,
автоматизація її обробки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри ІБтаН
Андрій КОРОТУН
“ ” 2024 року

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

БАЙШИ Вадима Костянтиновича
(ПРИЗВИЩЕ, ім'я, по батькові)

- Тема проєкту (роботи) Розробка захищеного P2P чату з шифруванням та автентифікацією повідомлень
Development of secure P2P chat with message encryption and authentication
керівник проєкту (роботи) к.т.н., доцент НЕЛАСА Ганна Вікторівна
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові.)
затверджені наказом закладу вищої освіти від “ 03 ” травня 2024 року № 190
- Строк подання студентом проєкту (роботи) 01 червня 2024 року
- Вихідні дані до проєкту (роботи) розглянути, проаналізувати та створити захищений P2P чат з шифруванням та автентифікації повідомлень
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Теоретичні основи алгоритму шифрування AES та підпис повідомлень алгоритмом ECDSA,
- Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) презентація у microsoft powerpoint (11 слайдів)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконання завдання
Розділи 1-3	НЕЛАСА Г.В., доцент каф. ІБтаН		
Нормоконтроль	КОРОЛЬКОВ Р. Ю., доцент каф. ІБтаН		

7. Дата видачі завдання » 01 » березня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Виконано
2	Аналіз літератури	1 тиждень	Виконано
3	Аналіз алгоритмів шифрування AES	2 тиждень	Виконано
4	Аналіз алгоритму підпису повідомлень ECDSA	3-4 тижні	Виконано
5	Розробка архітектури системи	6-7 тиждень	Виконано
6	Тестування та налагодження ПО	8 тиждень	Виконано
7	Нормоконтроль та рецензування.	9 тиждень	
8	Захист дипломної роботи.	10 тиждень	

Студент(ка)

(підпис)

Вадим БАЙША

(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)



(підпис)

Ганна НЕЛАСА

(Ім'я ПРИЗВИЩЕ)

АННОТАЦІЯ

Пояснювальна записка до дипломної роботи: 58 с., 22 рис., 17 джерел.

АВТЕНТИФІКАЦІЯ, МОДЕЛІ ЗАГРОЗ, ШИФРУВАННЯ
ПОВІДОМЛЕНЬ, ЦИФРОВИЙ ПІДПИС, AES, ECDSA, P2P .

Об'єкт дослідження - процеси захисту, автентифікації повідомлень та організації безпечного P2P зв'язку

Предмет дослідження - методи захисту та автентифікації повідомлень в P2P-чаті.

Мета дослідження - розробити захищений P2P чат з шифруванням та автентифікацією повідомлень для забезпечення конфіденційності та цілісності передачі даних.

Наукове та практичне значення результатів - дослідження та вдосконалення методів шифрування, автентифікації повідомлень та організації безпечного P2P зв'язку в чаті має наукове значення. Розроблений захищений P2P чат забезпечить конфіденційність та цілісність передачі даних, що має практичне значення для безпечного обміну повідомленнями в різних сферах діяльності.

ЗМІСТ

Перелік скорочень.....	6
Вступ.....	7
1 Основи побудови однорангових мереж	8
1.1 Огляд архітектури однорангових мереж без використання серверів	8
1.2 Аналіз протоколів для прямої взаємодії між клієнтами в P2P чатах.....	11
1.3 Методи шифрування повідомлень у P2P чатах	15
1.3.1 Симетричне шифрування.....	16
1.3.2 Асиметричне шифрування.....	18
1.3.3 Гібридне шифрування	20
1.4 Механізми автентифікації користувачів у P2P чатах.....	22
1.4.1 Криптографічні процеси в P2P чатах.....	24
1.4.2 Генерація ключів.....	26
1.5 Проблеми безпеки та конфіденційності у P2P чатах без використання серверів	29
1.6 Аналіз загроз та ризиків для конфіденційності, анонімності та безпеки користувачів у P2P чатах	31
1.7 Загрози конфіденційності та анонімності користувачів у P2P чатах....	36
2 Проектування захищеного P2P чату	38
2.1 Вибір технологій та інструментів для розробки.....	38
2.2 Вимоги до системи та функціональні специфікації	39
2.3 Архітектура захищеного P2P чату.....	39
2.4 Моделювання системи з використанням UML діаграм	40
2.5 Графічний інтерфейс користувача та обробка взаємодії з користувачем..	44
2.6 Висновки за результатами другого розділу.....	45
3 Реалізація та тестування захищеного P2P чату	46
3.1. Реалізація основних компонентів P2P чату	46
3.2 Інтеграція алгоритмів шифрування та аутентифікації	49
3.3 Тестування функціональності та безпеки P2P чату.....	52
Висновки.....	53
Перелік джерел посилань.....	57

ПЕРЕЛІК СКОРОЧЕНЬ

AES	- Advanced Encryption Standard (Розширений стандарт шифрування)
DDoS	- Denial-of-service (відмова в обслуговуванні)
DES	- Data encryption standard (Стандарт шифрування даних)
DTLS	- Datagram Transport Layer Security (Безпека транспортного рівня датаграми)
ECDSA	- Elliptic Curve Digital Signature Algorithm (Алгоритм цифрового підпису еліптичної кривої)
ICE	-Interactive Connectivity Establishment (Інтерактивне встановлення зв'язку)
MITM	- Man-in-the-Middle (людина всередині)
NaCl	- Networking and Cryptography library (Бібліотека мережевих технологій та криптографії)
P2P	- (Peer-to-Peer) від людини до людини
TLS	- Transport Layer Security (Безпека на транспортного рівня)
WebRTC	- Web Real-Time Communication (Інтернет-комунікація в реальному часі)

ВСТУП

В епоху стрімкого розвитку інформаційних технологій та збільшення обсягів даних, що передаються через мережу Інтернет, питання забезпечення безпечної та конфіденційної комунікації набуває особливої актуальності. Традиційні централізовані системи, такі як клієнт-серверна архітектура, мають певні обмеження щодо масштабованості, стійкості та конфіденційності. Саме тому децентралізовані P2P (Peer-to-Peer) мережі стають все більш популярними для побудови захищених та ефективних систем комунікації.

Одним із важливих застосувань P2P мереж є створення захищених чатів, які дозволяють користувачам здійснювати конфіденційне спілкування та обмін повідомленнями без необхідності залучення центральних серверів. Розробка таких систем вимагає застосування надійних методів шифрування та автентифікації повідомлень, щоб гарантувати приватність та цілісність комунікацій.

Метою даної дипломної роботи є розробка захищеного P2P чату з шифруванням та автентифікацією повідомлень. Основними завданнями є дослідження архітектури P2P мереж, аналіз існуючих протоколів та методів шифрування, проектування та реалізація системи захищеного чату, а також тестування її функціональності та безпеки.

1 ОСНОВИ ПОБУДОВИ ОДНОРАНГОВИХ МЕРЕЖ

1.1 Огляд архітектури однорангових мереж без використання серверів

В епоху стрімкого розвитку інформаційних технологій та зростання потреби у захищеній комунікації, особливої уваги заслуговують однорангові (peer-to-peer, P2P) мережі, які дозволяють забезпечити прямий обмін даними між користувачами без залучення централізованих серверів. Ця децентралізована архітектура відкриває нові можливості для створення захищених чатів, де конфіденційність та безпека передачі повідомлень є першочерговими завданнями.

Однорангові мережі без використання серверів базуються на принципі прямої взаємодії між клієнтами, де кожен вузол мережі виступає одночасно і клієнтом, і сервером. Це означає, що користувачі можуть безпосередньо обмінюватися повідомленнями та даними без необхідності проходження через центральний сервер, який би контролював та керував процесом комунікації. Така архітектура має низку переваг, серед яких можна виділити підвищену конфіденційність, відмовостійкість та масштабованість.

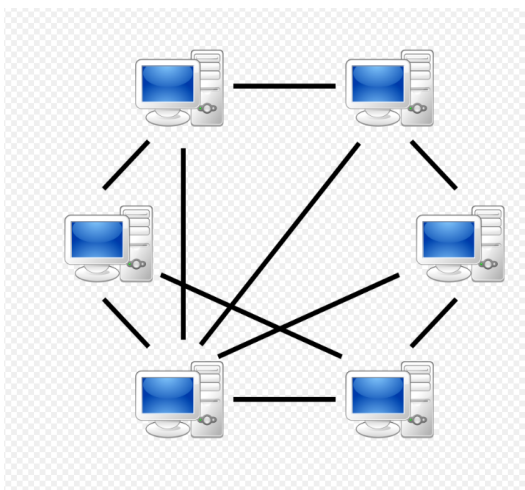


Рисунок 1.1 - Мережа peer to peer [1]

Відсутність центрального сервера в одноранговій мережі робить її більш стійкою до можливих збоїв та атак. У традиційній клієнт-серверній архітектурі

вихід з ладу центрального сервера може призвести до повної зупинки роботи всієї системи. Натомість, в одноранговій мережі без серверів, де кожен вузол є незалежним та самодостатнім, збій одного чи декількох вузлів не вплине на роботу інших частин мережі. Це забезпечує високу відмовостійкість та надійність системи в цілому.

Масштабованість є ще однією перевагою однорангових мереж без використання серверів. Завдяки прямій взаємодії між клієнтами, мережа може легко розширюватись та адаптуватись до зростаючої кількості користувачів. Кожен новий вузол, який приєднується до мережі, не лише користується її ресурсами, але й надає свої обчислювальні потужності та ресурси в розпорядження інших учасників. Це дозволяє ефективно розподіляти навантаження та підтримувати стабільну роботу системи незалежно від кількості користувачів.

Однак, однорангові мережі без серверів також мають свої виклики та проблеми, особливо у сфері безпеки та конфіденційності. Відсутність центрального контролю та управління може ускладнювати процес автентифікації користувачів та захисту від зловмисних дій. Крім того, прямий обмін даними між клієнтами вимагає надійних механізмів шифрування та збереження конфіденційності повідомлень. Забезпечення цілісності та автентичності даних також є важливим аспектом безпеки в одноранговій мережі.

Для вирішення цих проблем розробники захищених чатів на основі однорангових мереж без серверів повинні приділяти особливу увагу вибору відповідних протоколів та алгоритмів шифрування. Такі протоколи, як Off-the-Record (OTR) Messaging або Signal Protocol, можуть забезпечити надійне шифрування повідомлень та захист метаданих. Використання технологій асиметричного шифрування, таких як шифрування з відкритим ключем (Public Key Cryptography), дозволяє здійснювати безпечний обмін ключами та автентифікацію користувачів без необхідності довіри до централізованих серверів.

Ще одним важливим аспектом однорангових мереж без серверів є забезпечення прямої взаємодії між клієнтами. Для цього можуть використовуватися різні технології та протоколи, такі як WebRTC (Web Real-Time Communication), який дозволяє встановлювати прямі з'єднання між браузерами користувачів без необхідності встановлення додаткових плагінів або програмного забезпечення. Використання протоколу ICE (Interactive Connectivity Establishment) дозволяє уникати обмеження мережевих екранів (NAT) та забезпечувати прямий обмін даними між клієнтами.



Рисунок 1.2 - Принцип роботи NAT [2]

Отже, архітектура однорангових мереж без використання серверів відкриває нові можливості для створення захищених чатів, де конфіденційність та безпека комунікації є першочерговими завданнями. Прямий обмін даними між клієнтами, відмовостійкість та масштабованість є ключовими перевагами цієї архітектури. Однак, розробники повинні приділяти особливу увагу питанням безпеки, шифрування та автентифікації користувачів, щоб забезпечити надійний захист даних та конфіденційність спілкування в одноранговій мережі. Правильний вибір протоколів та алгоритмів, а також грамотна реалізація механізмів безпеки, є запорукою успішного створення захищеного чату на основі однорангової мережі без використання серверів.

1.2 Аналіз протоколів для прямої взаємодії між клієнтами в P2P чатах

У світі захищених P2P чатів вибір відповідного протоколу для прямої взаємодії між клієнтами є ключовим фактором, який впливає на ефективність, безпеку та конфіденційність комунікації. Протоколи визначають правила та стандарти, за якими відбувається обмін повідомленнями та даними між вузлами в мережі. Розглянемо детальніше деякі з найбільш поширених протоколів, які використовуються для прямої взаємодії між клієнтами в P2P чатах.

Одним з найвідоміших протоколів для прямої взаємодії в P2P мережах є протокол WebRTC (Web Real-Time Communication). WebRTC - це відкритий стандарт, який дозволяє веб-браузерам та мобільним додаткам встановлювати прямі з'єднання для обміну аудіо, відео та даними в реальному часі без необхідності встановлення додаткових плагінів. WebRTC використовує протокол ICE (Interactive Connectivity Establishment) для встановлення прямого з'єднання між клієнтами, обходячи обмеження мережевих екранів (NAT) та брандмауерів. Для передачі медіа-потоків WebRTC застосовує протокол SRTP (Secure Real-time Transport Protocol), який забезпечує шифрування та цілісність даних. Крім того, WebRTC підтримує шифрування сигнальних повідомлень за допомогою протоколу DTLS (Datagram Transport Layer Security). Завдяки своїй простоті та сумісності з веб-технологіями, WebRTC є популярним вибором для розробки захищених P2P чатів [3].

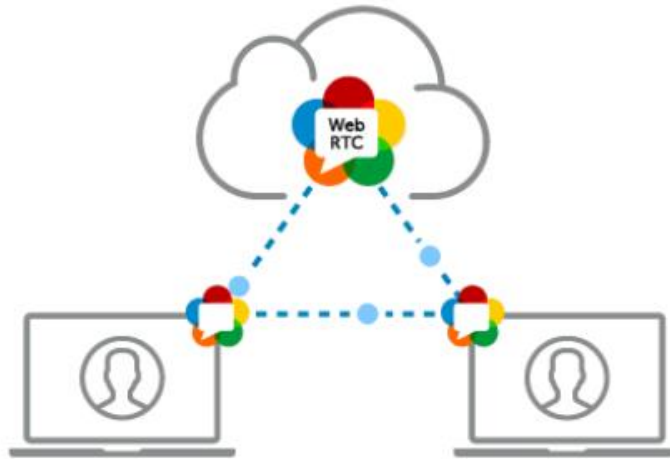


Рисунок 1.3 - Ілюстрація як працює WEB RTC [4]

Іншим протоколом, який заслуговує на увагу, є Matrix. Matrix - це відкритий протокол для децентралізованої комунікації в реальному часі. Він дозволяє створювати федеративні мережі серверів, які можуть взаємодіяти один з одним для передачі повідомлень. Хоча Matrix не є суто P2P протоколом, він підтримує прямі з'єднання між клієнтами за допомогою технології WebRTC. Matrix використовує протокол HTTP для передачі даних та забезпечує шифрування повідомлень за допомогою алгоритму Olm, який базується на протоколі Double Ratchet. Він також має вбудовані механізми автентифікації користувачів та керування доступом до кімнат (чатів). Matrix є гнучким та розширюваним протоколом, який дозволяє створювати різноманітні децентралізовані комунікаційні системи.

Протокол XMPP (Extensible Messaging and Presence Protocol) також може бути використаний для прямої взаємодії між клієнтами в P2P чатах. XMPP - це відкритий протокол для обміну повідомленнями та інформацією про присутність між клієнтами через децентралізовану мережу серверів. Хоча XMPP традиційно використовується в клієнт-серверних архітектурах, він також підтримує прямі з'єднання між клієнтами за допомогою розширення XMPP Extension Protocol (XEP-0174: Serverless Messaging). Для захисту комунікацій використовується потокове шифрування за допомогою протоколу TLS (Transport Layer Security) та автентифікацію користувачів через механізм

SASL (Simple Authentication and Security Layer). XMPP є розширюваним протоколом, що дозволяє додавати нові функції та можливості через використання додаткових розширень (XEP).

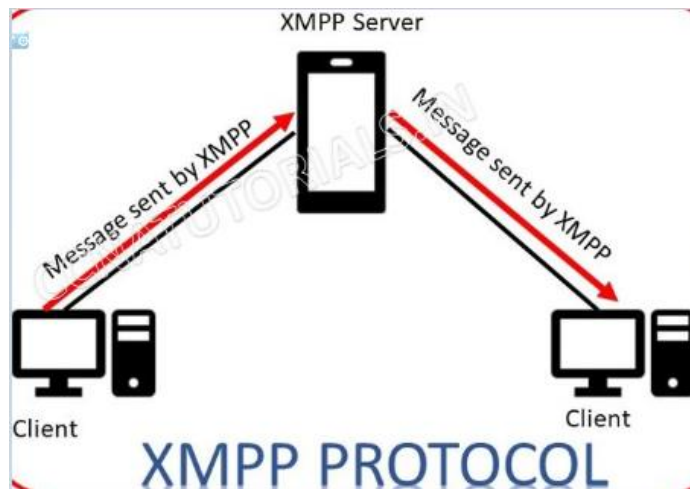


Рисунок 1.4 - Ілюстрація як працює XMPP протокол

Протокол Tox є ще одним цікавим рішенням для прямої взаємодії між клієнтами в P2P чатах. Tox - це відкритий протокол та набір інструментів для створення захищених і децентралізованих комунікаційних додатків. Він використовує розподілену хеш-таблицю (DHT) для встановлення прямих з'єднань між клієнтами та реалізує шифрування повідомлень за допомогою алгоритму NaCl (Networking and Cryptography library). Tox забезпечує наскрізне шифрування (end-to-end encryption) та підтримує різні типи комунікації, такі як текстові повідомлення, голосові та відеодзвінки. Однією з переваг Tox є його орієнтованість на збереження конфіденційності та анонімності користувачів.

Протокол Briar також заслуговує на увагу в контексті захищених P2P чатів. Briar - це захищений месенджер, який використовує однорангову мережу для прямої взаємодії між клієнтами. Він розроблений з акцентом на конфіденційність, безпеку та стійкість до цензури. Briar використовує комбінацію різних протоколів, таких як Tor (The Onion Router) для анонімізації трафіку, та протокол Double Ratchet для наскрізного шифрування повідомлень. Крім того, Briar підтримує синхронну та асинхронну комунікацію, що

дозволяє користувачам спілкуватися навіть в умовах нестабільного мережевого з'єднання.



Рисунок 1.5 - Ілюстрація як працює Bria протокол [5]

Вибір протоколу для прямої взаємодії між клієнтами в P2P чатах залежить від конкретних вимог та цілей проекту. Кожен протокол має свої переваги та особливості, які слід враховувати при розробці захищеного чату. Такі фактори, як рівень безпеки, конфіденційність, простота використання, масштабованість та сумісність з різними платформами та пристроями, повинні бути ретельно проаналізовані перед прийняттям рішення.

Важливо також розуміти, що протоколи для прямої взаємодії між клієнтами постійно розвиваються та вдосконалюються. Нові протоколи та розширення можуть з'являтися, пропонуючи покращені механізми безпеки та додаткові функціональні можливості. Тому при розробці захищеного P2P чату необхідно стежити за останніми тенденціями та дослідженнями в галузі мережевих протоколів та криптографії.

Окрім вибору протоколу, важливо також приділити увагу правильній реалізації та конфігурації обраного протоколу. Необхідно забезпечити безпечну генерацію ключів, управління ключами, автентифікацію користувачів та захист від різних типів атак. Регулярне оновлення та патчі безпеки також є критичними для підтримки високого рівня захищеності P2P чату.

Підсумовуючи, аналіз протоколів для прямої взаємодії між клієнтами в P2P чатах є важливим етапом у розробці захищеного та надійного рішення для спілкування. Розуміння переваг та особливостей кожного протоколу, а також ретельний аналіз вимог проекту, дозволяють обрати найбільш підходящий протокол та забезпечити високий рівень безпеки та конфіденційності комунікацій у децентралізованій мережі.

1.3 Методи шифрування повідомлень у P2P чатах

Методи шифрування повідомлень відіграють ключову роль у забезпеченні конфіденційності та безпеки комунікацій у P2P чатах. Вони дозволяють захистити вміст повідомлень від несанкціонованого доступу, перехоплення та розкриття третіми сторонами. У цьому пункті буде розглянуто основні методи шифрування, які використовуються в P2P чатах, їх особливості, переваги та недоліки. Одним з найпоширеніших методів шифрування в P2P чатах є симетричне шифрування.

Симетричне шифрування передбачає використання одного і того ж ключа для шифрування та дешифрування повідомлень. Перед початком спілкування відправник і одержувач повинні узгодити спільний секретний ключ, який буде використовуватися для захисту їхніх повідомлень.

Іншим поширеним методом шифрування в P2P чатах є асиметричне шифрування, також відоме як шифрування з відкритим ключем. У асиметричному шифруванні кожен учасник має пару ключів: відкритий ключ і приватний ключ. Відкритий ключ може вільно розповсюджуватися та використовуватися іншими учасниками для шифрування повідомлень, призначених для власника відповідного приватного ключа. Приватний ключ зберігається в таємниці і використовується для дешифрування отриманих повідомлень.

Гібридне шифрування є ще одним підходом, який поєднує переваги симетричного та асиметричного шифрування. У гібридних схемах шифрування повідомлення спочатку шифрується за допомогою випадково згенерованого симетричного ключа (ключа сеансу). Потім цей ключ сеансу шифрується за допомогою відкритого ключа одержувача. Зашифроване повідомлення та зашифрований ключ сеансу передаються одержувачу, який спочатку дешифрує ключ сеансу за допомогою свого приватного ключа, а потім використовує цей ключ для дешифрування самого повідомлення. Гібридне шифрування дозволяє використовувати ефективність симетричного шифрування для захисту великих обсягів даних, зберігаючи при цьому переваги асиметричного шифрування для безпечного обміну ключами.

Окрім вибору відповідного алгоритму шифрування, важливо також забезпечити правильну реалізацію та використання криптографічних примітивів. Безпечна генерація випадкових ключів, захист ключів від несанкціонованого доступу та регулярне оновлення ключів є критичними аспектами ефективного шифрування повідомлень у P2P чатах. Також важливо враховувати потенційні атаки на криптографічні системи, такі як атаки на основі шифротексту, атаки на основі відомого відкритого тексту або атаки "людина посередині" (Man-in-the-Middle), і вживати відповідних заходів для їх запобігання.

1.3.1 Симетричне шифрування

Симетричне шифрування використовує один і той самий ключ для шифрування та дешифрування повідомлень. Відправник і одержувач повинні заздалегідь обмінятися та узгодити спільний секретний ключ. Симетричне шифрування є ефективним для захисту великих обсягів даних і забезпечує високу швидкість роботи.



Рисунок 1.6 – Робота симетричної криптосистеми

Використовуючи шифрування із симетричними ключами може бути найефективнішим, отже користувачі не відчуватимуть жодної значної затримки в часі в результаті підчас шифрування та дешифрування. автентифікації використовує шифрування з симетричними ключами, що також забезпечує певний рівень безпеки, тому що інформація зашифрована одним симетричним ключем не може бути дешифрована яимось іншим симетричним ключем.

Таким чином, доки симетричний ключ зберігається в таємниці двома сторонами, що використовують його для шифрування обміну інформацією, то кожна сторона може бути впевнена, що обмінюється інформацією з іншою, так само як і до того часу доки дешифровані повідомлення матимуть зміст .

Приклади алгоритмів симетричного шифрування:

- AES (Advanced Encryption Standard): AES є одним із найпоширеніших алгоритмів симетричного шифрування. Він підтримує розміри ключів 128, 192 і 256 біт і забезпечує високий рівень безпеки. AES базується на принципі заміни та перестановки та виконує кілька раундів шифрування для захисту даних [6] .

- DES (Стандарт шифрування даних): DES - це симетричний алгоритм шифрування, розроблений у 1970-х роках і широко використовувався в минулому. DES використовує 56-бітний ключ і виконує 16 раундів шифрування на основі мережі Feistel. Однак через відносно короткий розмір ключа DES вважається застарілим і вразливим до сучасних методів

криптоаналізу. Рекомендується використовувати більш надійний алгоритм, наприклад AES [6] .

Це один із перших широко використовуваних алгоритмів симетричного шифрування. Він був розроблений компанією IBM у 1970-х роках і прийнятий урядом США як стандарт шифрування. DES використовував 56-бітний ключ, який вважається відносно коротким за сучасними стандартами.

Алгоритм DES заснований на мережі Фейстеля, яка розділяє вхідні дані на дві половини та виконує серію перестановок і замін на них у кожному раунді шифрування. DES виконує 16 раундів шифрування, кожен з яких використовує окремий 48-бітний підключ, отриманий з вихідного 56-бітного ключа [6] .

1.3.2 Асиметричне шифрування

Асиметричне шифрування, також відоме як шифрування з відкритим ключем, використовує пару ключів: відкритий ключ і приватний ключ. Відкритий ключ може вільно розповсюджуватися та використовуватися будь-ким для шифрування повідомлень, тоді як приватний ключ зберігається в таємниці і використовується для дешифрування повідомлень. Асиметричне шифрування забезпечує більшу гнучкість і спрощує процес обміну ключами порівняно з симетричним шифруванням.

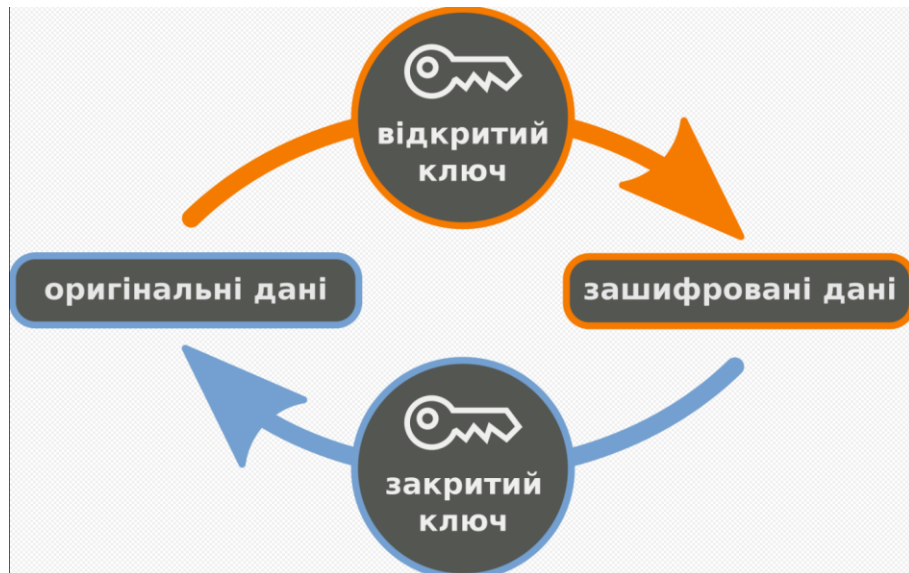


Рисунок 1.7 – Робота асиметричної криптосистеми [7]

Приклади алгоритмів асиметричного шифрування: RSA (Rivest-Shamir-Adleman):

- RSA є одним з найбільш широко використовуваних алгоритмів асиметричного шифрування. Він був розроблений в 1977 році Рональдом Рівестом, Аді Шаміром і Леонардом Адлеманом. RSA базується на складності факторизації великих цілих чисел. Ключова пара RSA складається з відкритого ключа (n, e) і приватного ключа (n, d) , де n є добутком двох великих простих чисел, а e і d є взаємно простими числами, такими що $(e * d) \bmod (p-1)(q-1) = 1$. Відкритий ключ використовується для шифрування повідомлень, а приватний ключ - для дешифрування. RSA забезпечує високий рівень безпеки, але є відносно повільним порівняно з симетричними алгоритмами. Він широко використовується для шифрування, цифрового підпису та обміну ключами [6].

- DSA (Digital Signature Algorithm): DSA є алгоритмом цифрового підпису, який був розроблений Національним інститутом стандартів і технологій США (NIST) в 1991 році. DSA використовується для створення та перевірки цифрових підписів, але не для шифрування даних. Алгоритм базується на складності обчислення дискретних логарифмів в скінченному полі. Ключова пара DSA складається з відкритого ключа (p, q, g, y) і

приватного ключа (x), де p є великим простим числом, q - дільником $p-1$, g - генератором підгрупи порядку q в полі F_p , y - відкритим ключем, обчисленим як $y = g^x \pmod{p}$, а x - приватним ключем. Для створення цифрового підпису повідомлення спочатку обчислюється його хеш-значення, а потім підпис генерується з використанням приватного ключа та хеш-значення. Перевірка підпису здійснюється з використанням відкритого ключа, хеш-значення повідомлення та самого підпису. DSA забезпечує цілісність і автентичність повідомлень, але має певні обмеження щодо розміру повідомлень і швидкості роботи порівняно з іншими алгоритмами цифрового підпису, такими як ECDSA [6].

Варто зазначити, що RSA і DSA мають різне призначення. RSA використовується як для шифрування, так і для цифрового підпису, тоді як DSA призначений виключно для цифрового підпису. Вибір між RSA і DSA залежить від конкретних вимог безпеки, продуктивності та сумісності в контексті P2P чату або іншої системи захищеної комунікації.

1.3.3 Гібридне шифрування

Гібридне шифрування складається з обох видів шифрування, асиметричне та симетричне. Тобто він використовує як прямі, так і непрямі методи шифрування. Зокрема, відзначаються основні переваги обох методів.

Гібридний криптографічний метод використовується в ліцензіях SSL/TLS під час зв'язку між серверами та клієнтами (веб-браузерами), який відомий як «рукописання TLS». По-перше, легітимність кожної сторони оцінюється за допомогою закритого та відкритого ключів. Після того, як обидві сторони визначили свої особи, дані шифруються за допомогою симетричного методу, який використовує ефемерний (сеансовий) ключ. Це гарантує швидку передачу великого обсягу даних, які ми надсилаємо та

отримуємо онлайн щохвилини. Його алгоритм в кінцевому підсумку дає наступну послідовність дій: сторона X створює відкритий і секретний пару ключів. Відкритий ключ передається по вразливому каналу стороні Y, яка повинна надати секретну інформацію стороні X [8].

Сторона Y генерує випадковий пароль і використовує симетричний алгоритм шифрування для перезапису інформації. Використовуючи відкритий ключ від Сторона X отримує секретну інформацію від Сторони Y та ініціює асиметричний метод. Симетричний ключ виводиться з отриманого повідомлення та розшифровується за допомогою закритого ключа. Решта інформації розшифровується за допомогою симетричного алгоритму, який використовує спільний ключ дешифрування між відправником і одержувачем. Це дозволяє досягти прийнятної швидкості обробки даних при використанні блочно-симетричних алгоритмів і вирішує проблему передачі ключів по каналу.

Симетричні алгоритми, як правило, дуже надійні, але іноді можуть бути скомпрометовані, як-от помилка 1993 року в алгоритмі DES національної безпеки США. Таким чином, немає жодної гарантії, що нові досягнення в криптоаналізі є постійними, що сфера обчислювальної техніки продовжує вдосконалюватися. Можливості комп'ютера не в змозі виявити недоліки в симетричному алгоритмі, який дозволяє третім особам розшифровувати незахищені дані. Щоб вирішити цю проблему, вони продовжують вдосконалювати алгоритм і збільшувати довжину ключа. Удосконалення алгоритмів не завжди вирішує проблему. У майбутньому, коли буде доступно більше технічних засобів, проблема буде вирішена.

Для вирішення згаданих вище проблем було розроблено метод захисту даних, який використовує багаторівневе шифрування та дешифрування даних.

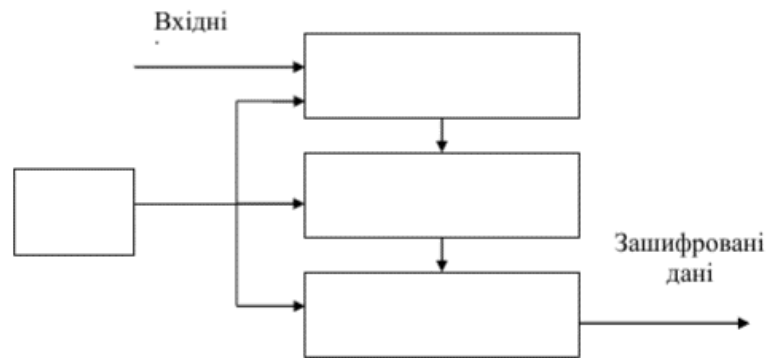


Рисунок 1.8 – Ілюстрація того, як працює гібридне шифрування

Унікальність методу полягає в тому, що задіяні всі перераховані вище алгоритми складної обробки даних: поряд з асиметричним алгоритмом використовується кілька інших симетричних алгоритмів, кожен з яких застосовується послідовно, шар за шаром.

У результаті, якщо один із симетричних алгоритмів зламано, дані будуть захищені іншим. З точки зору швидкості, це найбільш ефективно використовується.

1.4 Механізми автентифікації користувачів у P2P чатах

Механізми автентифікації користувачів відіграють ключову роль у забезпеченні безпеки та конфіденційності P2P чатів. Вони гарантують, що лише авторизовані користувачі можуть приєднатися до чату та брати участь у спілкуванні. Давайте розглянемо основні механізми автентифікації користувачів у P2P чатах.

Доволі популярним механізмом автентифікації - це використання цифрових підписів. Кожне повідомлення, надіслане користувачем, підписується його приватним ключем. Інші учасники чату можуть перевірити цифровий підпис за допомогою публічного ключа відправника. Якщо підпис дійсний, це гарантує, що повідомлення справді надіслане заявленим

відправником і не було змінено під час передачі. Цифрові підписи забезпечують автентичність та цілісність повідомлень.

Також одним із методів автентифікації користувачів - це використання сертифікатів. Сертифікат - це цифровий документ, який засвідчує зв'язок між публічним ключем та особистістю користувача. Сертифікати можуть видаватися довіреними центрами сертифікації. Коли користувач приєднується до P2P чату, він надає свій сертифікат іншим учасникам. У деяких P2P чатах використовуються децентралізовані механізми автентифікації, такі як блокчейн або розподілені системи довіри. У цих випадках автентифікація користувачів здійснюється на основі консенсусу між учасниками мережі [8].

Наприклад, у блокчейн-системах користувачі можуть реєструвати свої публічні ключі в розподіленому реєстрі, який підтримується та перевіряється всіма вузлами мережі. Коли користувач приєднується до чату, його особистість може бути підтверджена шляхом перевірки відповідності його публічного ключа з інформацією, збереженою в блокчейні. Для посилення безпеки автентифікації в P2P чатах можуть використовуватися додаткові фактори, такі як паролі, одноразові коди або біометричні дані. Окрім перевірки цифрового підпису, користувачу може бути запропоновано ввести пароль або назвати ідентифікаційне слово. Ці додаткові фактори автентифікації ускладнюють несанкціонований доступ до чату, навіть якщо приватний ключ користувача скомпрометований.

Важливо також враховувати питання конфіденційності при реалізації механізмів автентифікації в P2P чатах. Деякі користувачі можуть надавати перевагу анонімності та не бажати розкривати свою особисту інформацію. У таких випадках можуть використовуватися псевдонімні або анонімні механізми автентифікації, які дозволяють користувачам підтвердити свою автентичність без розкриття їхньої реальної особистості. Незалежно від обраного механізму автентифікації, важливо забезпечити надійне управління ключами та сертифікатами. Користувачі повинні зберігати свої приватні ключі

в безпечному місці та регулярно оновлювати їх. У разі компрометації ключів або сертифікатів, вони повинні бути негайно відкликані та замінені новими.

Також важливо враховувати можливість відкликання доступу для користувачів, які порушують правила чату або демонструють зловмисну поведінку. Механізми відкликання доступу дозволяють адміністраторам чату або довіреним учасникам позбавляти недобросовісних користувачів можливості приєднуватися до чату та брати участь у спілкуванні. Загалом, механізми автентифікації користувачів у P2P чатах є невід'ємною частиною забезпечення безпеки та конфіденційності спілкування. Вони гарантують, що лише авторизовані користувачі можуть приєднатися до чату та що повідомлення надходять від достовірних відправників. Вибір відповідного механізму автентифікації залежить від конкретних вимог та особливостей P2P чату, а також від балансу між безпекою та зручністю використання. Ретельна реалізація та управління механізмами автентифікації є ключовими факторами для підтримки довіри та захисту конфіденційності користувачів у P2P чатах [8].

1.4.1 Криптографічні процеси в P2P чатах

Безпека та конфіденційність комунікацій виходять на перший план. Ніхто не хоче, щоб його приватні розмови потрапили в чужі руки або були скомпрометовані. Саме тут на допомогу приходять криптографічні процеси. Вони забезпечують надійний захист повідомлень та гарантують, що тільки авторизовані учасники чату зможуть їх прочитати.

По-перше, кожен учасник P2P-чату повинен згенерувати власну пару ключів - відкритий ключ і закритий ключ. Це як два унікальні замки, які захищають ваші комунікації. Відкритий ключ можна вільно поширювати серед інших учасників чату, як візитну картку. Але закритий ключ — це

таємниця, і її потрібно ретельно зберігати. Він використовується для розшифровки повідомлень, які були зашифровані за допомогою вашого відкритого ключа. Після обміну відкритими ключами учасники чату можуть встановити безпечне з'єднання. Це як будувати надійний міст довіри між вами та вашим співрозмовником. Протоколи обміну ключами, такі як Diffie-Hellman, дозволяють учасникам створювати спільний секрет, навіть якщо вони спілкуються через незахищений канал зв'язку. Це як погодитися на пароль, який розумієте лише ви двоє.

Тепер, коли в учасників є спільний секретний ключ, вони можуть почати обмінюватись зашифрованими повідомленнями. Перед відправкою повідомлення шифрується за допомогою узгодженого алгоритму, наприклад, AES. Це як загорнути ваше повідомлення в непрозорий конверт, який може відкрити лише одержувач зі своїм секретним ключем. Навіть якщо хтось перехопить ваше повідомлення, він не зможе його прочитати без відповідного ключа. Але як бути впевненим, що повідомлення надійшло саме від того, хто його відправив? Тут на допомогу приходять цифрові підписи. Відправник створює унікальний цифровий підпис для кожного повідомлення, використовуючи свій приватний ключ. Це як поставити свій унікальний відбиток пальця на повідомленні. Одержувач може перевірити підпис за допомогою відкритого ключа відправника і переконатися, що повідомлення справді надійшло від нього і не було змінено під час передачі.

Управління ключами - це ще один важливий аспект криптографічних процесів у P2P чатах. Ключі потрібно зберігати в безпечному місці, регулярно оновлювати і, якщо потрібно, відкликати. Це як стежити за своїми ключами від будинку - потрібно бути обережним, щоб вони не потрапили в чужі руки і завжди мати запасні на випадок, якщо основні будуть втрачені або скомпрометовані. У P2P чатах також важливо забезпечити пряме з'єднання між учасниками. Це як розмовляти віч-на-віч, без посередників. Протоколи, такі як ICE і STUN, допомагають встановити пряме з'єднання, навіть якщо учасники знаходяться за мережевими екранами або у різних мережах. [8].

Криптографічні процеси в P2P чатах - це щит, який захищає вашу конфіденційність та цілісність комунікацій. Вони гарантують, що лише авторизовані учасники можуть читати ваші повідомлення і що ніхто не зможе їх змінити непомітно. Це як мати свій власний секретний код, який розуміють лише ви та ваші довірені співрозмовники. Звісно, як і будь-яка система безпеки, криптографічні процеси в P2P чатах повинні бути ретельно розроблені та реалізовані. Розробники повинні використовувати надійні криптографічні алгоритми, регулярно оновлювати бібліотеки та проводити ретельні аудити безпеки. Це як будувати міцний замок для ваших комунікацій - потрібно використовувати найкращі матеріали та регулярно перевіряти, чи немає слабких місць.

Отже, криптографічні процеси в P2P чатах - це невидимі захисники приватності та безпеки. Вони працюють за лаштунками, забезпечуючи конфіденційність, цілісність та автентичність ваших розмов. Завдяки ним ви можете спілкуватися вільно та безпечно, знаючи, що ваші повідомлення захищені надійним криптографічним щитом. Тож, коли ви користуєтесь P2P чатом з криптографічними процесами, ви можете бути впевнені, що ваші розмови залишаться приватними та захищеними від сторонніх очей.

1.4.2 Генерація ключів

Генерація ключів є фундаментальним процесом в криптографії, який лежить в основі безпечної комунікації в P2P чатах. Метою генерації ключів є створення унікальних криптографічних ключів для кожного учасника чату. Ці ключі використовуються для шифрування, дешифрування, підпису та перевірки автентичності повідомлень.

Типи ключів:

- симетричні ключі: використовуються в алгоритмах симетричного шифрування, таких як AES. Один і той самий ключ використовується для шифрування та дешифрування повідомлень.

- асиметричні ключі: використовуються в алгоритмах асиметричного шифрування, таких як RSA. Кожен учасник має пару ключів - відкритий ключ і приватний ключ. Відкритий ключ використовується для шифрування повідомлень та перевірки цифрових підписів, а приватний ключ - для дешифрування повідомлень та створення цифрових підписів.

Процес генерації ключів.

1. Вибір алгоритму: вибирається криптографічний алгоритм для генерації ключів, наприклад ECDSA.

ECDSA (Elliptic Curve Digital Signature Algorithm): ECDSA є варіантом DSA, який базується на еліптичних кривих.

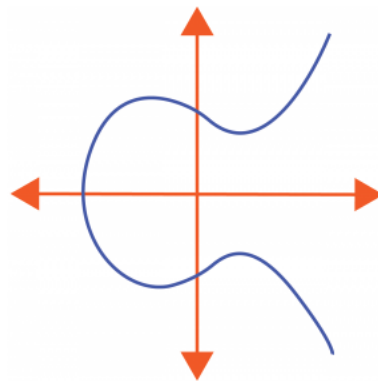


Рисунок 1.9 – Еліптична крива [9]

Генерація ключів ECDSA.

Вибирається еліптична крива E над скінченним полем $\mathbb{F}_p$ (де p - просте число) або $\mathbb{F}_{2^m}$ (де m - ціле число).

Вибирається базова точка G на еліптичній кривій.

Вибирається випадкове ціле число d , таке що $0 < d < n$, де n - порядок точки G . Число d є приватним ключем.

Обчислюється відкритий ключ $Q = d * G$ (множення точки на скаляр).

Відкритий ключ складається з параметрів еліптичної кривої (E, G, n) та точки Q , а приватний ключ - з числа d .

ECDSA забезпечує високий рівень безпеки з меншими розмірами ключів порівняно з RSA та DSA.

2. Генерація випадкових чисел: генеруються випадкові числа з використанням криптографічно безпечного генератора випадкових чисел (CSPRNG). Ці числа слугують основою для створення ключів.

3. Генерація ключів: на основі випадкових чисел та обраного алгоритму генеруються криптографічні ключі. Для асиметричних алгоритмів генерується пара ключів - відкритий ключ і приватний ключ.

4. Перевірка якості ключів: згенеровані ключі перевіряються на відповідність певним критеріям якості, таким як достатня ентропія, унікальність та відповідність стандартам.

5. Збереження ключів: згенеровані ключі зберігаються в захищеному сховищі, наприклад, в зашифрованому файлі або на апаратному модулі безпеки (HSM).

Вимоги до генерації ключів.

Випадковість - ключі повинні генеруватися з використанням надійних джерел ентропії та криптографічно безпечних генераторів випадкових чисел, щоб забезпечити їх непередбачуваність та унікальність.

Достатня довжина ключів - довжина ключів повинна бути достатньою для забезпечення криптографічної стійкості. Наприклад, рекомендована довжина ключів RSA становить щонайменше 2048 біт, а для ECDSA - 256 біт. Конфіденційність: процес генерації ключів повинен відбуватися в довіреному середовищі, і згенеровані ключі повинні зберігатися в захищеному вигляді, щоб запобігти їх компрометації.

Регулярне оновлення - ключі повинні періодично оновлюватися, щоб зменшити ризики, пов'язані з їх можливою компрометацією. Період оновлення залежить від специфіки системи та вимог безпеки.

Управління ключами.

Безпечне зберігання - згенеровані ключі повинні зберігатися в захищеному сховищі, наприклад, в зашифрованому файлі або на апаратному модулі безпеки (HSM), щоб запобігти несанкціонованому доступу.

Резервне копіювання - ключі повинні бути надійно збережені та мати резервні копії на випадок втрати або пошкодження.

Розповсюдження ключів - відкриті ключі можуть вільно розповсюджуватися між учасниками чату, тоді як приватні ключі повинні зберігатися в таємниці та передаватися лише через захищені канали.

Відкликання ключів - у разі компрометації ключа або завершення строку його дії, ключ повинен бути відкликаний, а учасники чату повинні бути сповіщені про необхідність оновлення ключів [10].

Генерація ключів є критично важливим процесом для забезпечення безпеки P2P чатів. Правильна генерація, зберігання та управління ключами є запорукою конфіденційності, цілісності та автентичності комунікацій. Розробники повинні слідувати сучасним стандартам та рекомендаціям щодо генерації ключів, використовувати перевірені криптографічні бібліотеки та забезпечувати належний захист ключів протягом усього їх життєвого циклу.

1.5 Проблеми безпеки та конфіденційності у P2P чатах без використання серверів

P2P чати без використання серверів мають свої унікальні проблеми безпеки та конфіденційності, які необхідно враховувати при розробці та використанні таких систем. Відсутність центрального сервера, який би контролював та захищав комунікації, створює додаткові виклики для забезпечення безпеки та конфіденційності. Розглянемо основні проблеми безпеки та конфіденційності у P2P чатах без використання серверів.

1. Захист від атак "людина посередині" (Man-in-the-Middle attacks):

- для захисту від таких атак необхідно використовувати надійні методи шифрування та автентифікації, такі як наскрізне шифрування (end-to-end encryption) та цифрові підписи.

2. Забезпечення конфіденційності та анонімності учасників:

- у P2P чатах без серверів важче забезпечити повну анонімність учасників, оскільки їх IP-адреси та інша ідентифікуюча інформація можуть бути доступні іншим учасникам.

- для захисту конфіденційності та анонімності учасників можуть використовуватися різні методи, такі як використання псевдонімів, шифрування метаданих та технології анонізації (наприклад, Tor або I2P) [11].

3. Захист від атак Сібіли (Sybil attacks):

- у P2P чатах без серверів зловмисник може створювати multiple ідентичності (так звані "сібіли") для впливу на мережу або отримання контролю над нею.

- для захисту від атак Сібіли можуть використовуватися механізми репутації, верифікації учасників або алгоритми консенсусу, які ускладнюють створення та підтримку фальшивих ідентичностей.

4. Забезпечення цілісності та автентичності повідомлень:

- у P2P чатах без серверів важливо гарантувати, що повідомлення не були змінені або підроблені під час передачі.

- для забезпечення цілісності та автентичності повідомлень використовуються криптографічні методи, такі як цифрові підписи та хеш-функції.

- кожне повідомлення повинно бути підписане відправником за допомогою його приватного ключа, що дозволяє одержувачам перевірити автентичність повідомлення [11].

1.6 Аналіз загроз та ризиків для конфіденційності, анонімності та безпеки користувачів у P2P чатах

Однією з основних загроз у P2P чатах є атаки "людина посередині" (Man-in-the-Middle, MITM). У цьому типі атаки зловмисник перехоплює комунікацію між двома учасниками чату і може читати, змінювати або підробляти повідомлення. Зловмисник може видавати себе за легітимного учасника чату, перенаправляючи повідомлення через свій пристрій. Це дозволяє йому отримати доступ до конфіденційної інформації та порушити цілісність спілкування. Для захисту від атак MITM важливо використовувати надійні механізми шифрування, такі як наскрізне шифрування (end-to-end encryption), та перевіряти автентичність учасників за допомогою цифрових підписів або сертифікатів [12].

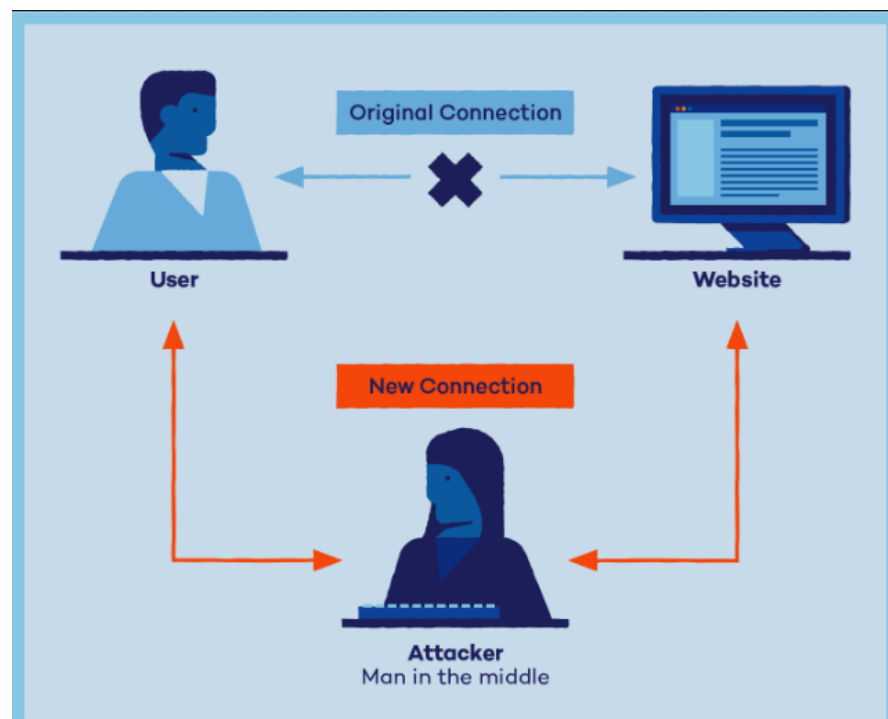


Рисунок 1.10 - Принцип атаки Man-in-the-Middle [13]

Інша поширена вразливість у P2P чатах - це атаки Сібіли (Sybil attacks). У цьому типі атаки зловмисник створює велику кількість фальшивих

ідентичностей (сібіл) і використовує їх для отримання непропорційного впливу в мережі. Зловмисник може використовувати ці фальшиві ідентичності для поширення спаму, здійснення DoS-атак або маніпулювання консенсусом у мережі. Для захисту від атак Сібіли P2P чати можуть використовувати механізми репутації, верифікації ідентичності або алгоритми консенсусу, які ускладнюють створення та підтримку великої кількості фальшивих вузлів.

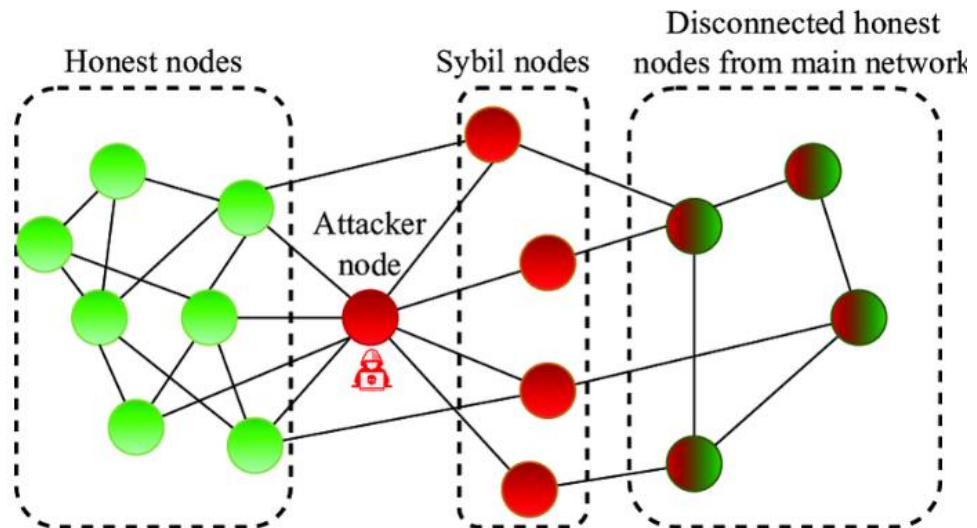


Рисунок 1.11 – Ілюстрація того як працює Sybil attacks [14]

Ще однією загрозою для P2P чатів є атаки на конфіденційність, такі як аналіз трафіку або розкриття метаданих. Зловмисник може спостерігати за мережевим трафіком і аналізувати шаблони комунікації, щоб отримати інформацію про учасників чату, їхні взаємозв'язки або вміст повідомлень. Навіть якщо самі повідомлення зашифровані, метадані, такі як IP-адреси, часові позначки або розмір повідомлень, можуть розкрити конфіденційну інформацію. Для захисту від атак на конфіденційність P2P чати можуть використовувати техніки анонімізації, такі як маршрутизація цибулі (onion routing) або перемішування (mixing), а також застосовувати механізми приховування метаданих.

Вразливості програмного забезпечення можуть становити серйозну загрозу для безпеки P2P чатів. Недоліки в реалізації протоколів, криптографічних бібліотеках або клієнтському програмному забезпеченні

можуть призвести до появи вразливостей, які зловмисники здатні використати для несанкціонованого доступу, виконання шкідливого коду або витоку конфіденційних даних. Щоб виявити та усунути ці вразливості, необхідно регулярно оновлювати програмне забезпечення, використовувати надійні бібліотеки та проводити ретельні аудити безпеки.

Однією з найбільших загроз є розкриття IP-адреси. IP-адреса - це унікальний числовий ідентифікатор, присвоєний кожному пристрою в мережі Інтернет, який використовується для ідентифікації та адресації пристроїв, забезпечуючи можливість обміну даними між ними. У P2P чатах IP-адреси учасників можуть бути видимі для інших вузлів мережі через прямі з'єднання між користувачами. Зловмисник може отримати IP-адресу користувача різними способами, наприклад, якщо чат використовує прямі з'єднання між користувачами без проміжних серверів, або шляхом перехоплення трафіку чи використання вразливостей у протоколі P2P.

Розкриття IP-адреси може мати серйозні наслідки для конфіденційності користувача. За допомогою IP-адреси зловмисник може визначити географічне розташування, провайдера інтернет-послуг, а іноді навіть особисту інформацію користувача. Це може призвести до таргетованих атак, спаму, шантажу та інших зловмисних дій. Реальні випадки порушення конфіденційності через розкриття IP-адрес у P2P мережах включають деанонімізацію користувачів торент-трекерів, викриття особистостей учасників анонімних мереж та ідентифікацію відправників біткоїн-транзакцій через аналіз вхідних вузлів мережі [15].

Аналіз трафіку тісно пов'язаний з IP-адресою, оскільки вона є ключовим елементом, який дозволяє зловмисникам здійснювати аналіз трафіку та отримувати цінну інформацію про користувачів та їх поведінку в мережі. Кожен пакет даних містить IP-адреси відправника та одержувача, які використовуються для маршрутизації пакетів між вузлами мережі. Однак наявність IP-адрес у пакетах даних створює значні ризики для конфіденційності.

Зловмисник, який має доступ до мережевої інфраструктури або може перехоплювати трафік, може легко отримати IP-адреси учасників P2P чату та здійснювати аналіз трафіку, вивчаючи шаблони комунікації, частоту та обсяги переданих даних, а також інші метадані. Це дозволяє зловмиснику ідентифікувати учасників чату, відстежувати їх активність, визначати географічне розташування та здійснювати таргетовані атаки [15].

Аналіз трафіку - це процес перехоплення, запису та аналізу мережових пакетів з метою отримання інформації про поведінку користувачів, типи даних, що передаються, та шаблони комунікації. У P2P чатах аналіз трафіку може бути використаний зловмисниками для збору метаданих про користувачів та їх активність, навіть якщо зміст повідомлень зашифрований. Метадані, які можуть бути зібрані, включають час та частоту відправки повідомлень, розмір повідомлень, IP-адреси відправника та одержувача, тривалість сесій.

Зловмисник може використовувати ці метадані для створення деталізованих профілів користувачів, відстеження їх активності та взаємодій, ідентифікації ключових учасників мережі або навіть спроб дешифрування змісту повідомлень на основі статистичного аналізу. Інструменти та методи для аналізу трафіку включають сніфери пакетів, статистичний аналіз та машинне навчання.

P2P чати також можуть бути вразливими до атак на доступність, таких як розподілені атаки типу "відмова в обслуговуванні" (DDoS), коли зловмисник перевантажує мережу або окремі вузли великою кількістю запитів, порушуючи нормальне функціонування чату. Для захисту від DDoS-атак можуть використовуватися механізми розподілу навантаження, фільтрації трафіку або репутаційні системи. Соціальна інженерія та фішинг також становлять загрозу для користувачів P2P чатів.



Рисунок 1.12 - Соц. інженерія [16]

Зловмисники можуть намагатися обманом змусити користувачів розкрити свої приватні ключі паролі або іншу конфіденційну інформацію, використовуючи фішингові повідомлення, підроблені веб-сайти або шкідливі посилання.

Освіта користувачів щодо належних практик безпеки є важливою для захисту від атак соціальної інженерії. Для ефективного аналізу загроз та вразливостей у P2P чатах необхідно проводити регулярні оцінки ризиків і тести на проникнення.



Рисунок 1.13 - Фішинг [17]

Це дозволяє виявити потенційні слабкі місця в системі безпеки, оцінити ймовірність і вплив різних типів атак, а також розробити відповідні стратегії пом'якшення ризиків. Моніторинг мережевої активності, ведення журналів подій і використання систем виявлення вторгнень можуть допомогти вчасно виявити та реагувати на потенційні загрози. Важливо також враховувати юридичні та регуляторні аспекти безпеки P2P чатів. Залежно від юрисдикції

та типу даних, що передаються, можуть існувати вимоги щодо захисту конфіденційності, зберігання даних або розкриття інформації правоохоронним органам.

Розробники та адміністратори P2P чатів повинні бути обізнані з відповідними законами та нормативними актами й забезпечувати їх дотримання. Загалом, аналіз загроз та вразливостей у P2P чатах є складним і багатогранним процесом. Він вимагає комплексного підходу, що враховує технічні, соціальні та правові аспекти безпеки. Розробники та адміністратори P2P чатів повинні приділяти особливу увагу забезпеченню конфіденційності, цілісності та доступності комунікацій, використовуючи надійні механізми шифрування, автентифікації та захисту від різних типів атак. Регулярні оновлення, моніторинг і аудит безпеки є невід'ємними складовими підтримки безпечного середовища для спілкування в P2P чатах.

1.7 Загрози конфіденційності та анонімності користувачів у P2P чатах

Приєднуючись до P2P-чату для веб-спілкування з друзями та однодумцями, користувачі часто розраховують на збереження конфіденційності та анонімності, адже особисте життя не повинно бути предметом загального огляду. Однак навіть у середовищі P2P-чатів певні ризики та загроза конфіденційності та анонімності користувачів.

Однією з основних проблем є можливість розкриття IP-адреси. У мережах P2P кожен вузол (або кожен учасник чату) ідентифікує унікальну IP-адресу. Якщо зломисник отримав доступ до IP-адреси користувача, він може спробувати встановити його сайт, ідентифікувати особу та відстежити онлайн-активність. Це стає значною загрозою приватності користувачів.

Крім того, якщо IP-адреса користувача залишається прихованою, інші методи отримують додаткову інформацію про нього, зокрема, шляхом аналізу

мережевого трафіку. Зловмисник може перехоплювати та аналізувати шаблони комунікації користувача, такі як частота спілкування, часові проміжки активності та список контактів. Ці метадані можуть розкрити інформацію про звички та інтереси користувача, навіть якщо вміст повідомлення зашифрований.

Особливу загрозу становлять атаки на анонімність, коли зловмисник намагається розкрити справжню особистість користувача шляхом розміщення його дій у P2P-чаті з іншою публічно доступною інформацією в Інтернеті. Це може призвести до деанонізації користувача та порушення його приватності.

Різні технічні аспекти, загрози можуть виникнути і через довірливість користувачів до своїх співрозмовників у P2P-чаті. Якщо користувач ділиться конфіденційною інформацією з нібито довіреною особою, яка згодом виявляється зловмисником, це може призвести до розголошення приватних даних. Крім того, зловмисники можуть вдаватися до методів соціальної інженерії, щоб маніпулювати користувачами та отримувати доступ до чутливої інформації.

Для вирішення цих проблем розробники P2P-чатів включають різноманітні механізми безпеки та конфіденційності. Серед них можна виділити наскрізне шифрування (наскрізне шифрування), яке забезпечує доступ до вмісту, повідомляє лише відправнику та одержувачу, а також протоколи анонізації, які підтримують реальні IP-адреси користувачів. Використання захищених каналів передачі даних забезпечує також рівень безпеки комунікацій.

Таким чином, незважаючи на переваги P2P-чатів у плані приватності та децентралізації, вони все ще забезпечать певні ризики для конфіденційності та анонімності користувачів. Розкриття IP-адреси, аналіз трафіку, атаки на анонімність та загрози, пов'язані з людським фактором, вимагають комплексного підходу до забезпечення безпеки.

2 ПРОЕКТУВАННЯ ЗАХИЩЕНОГО P2P-ЧАТУ

2.1 Вибір технологій та інструментів для розробки

Для розробки захищеного P2P-чату було обрано мову програмування Java, яка є надійною, кросплатформенною та широко використовуваною мовою та надає потужні інструменти для розробки мережних додатків і реалізації криптографічних функцій. Також існує велика екосистема бібліотек та фреймворків, що полегшують розробку та забезпечують необхідний функціонал.

Для реалізації криптографічних функцій, таких як шифрування та цифрові підписи, було використано наступні бібліотеки:

- `java.security.*` та `java.security.spec.*` - для роботи з криптографічними алгоритмами та специфікаціями ключів.
- `javax.crypto.*` та `javax.crypto.spec.SecretKeySpec` - для реалізації симетричного шифрування за допомогою алгоритму AES.
- `javax.crypto.Cipher` та `javax.crypto.spec.SecretKeySpec` - для шифрування повідомлень з використанням AES.
- `java.security.*` та `java.security.spec.ECGenParameterSpec` - для генерації ключів та реалізації цифрового підпису за допомогою алгоритму ECDSA.
- `java.util.Base64` - для кодування та декодування ключів та підписів.

Ці бібліотеки надають необхідні класи та методи для виконання криптографічних операцій, таких як шифрування повідомлень симетричним алгоритмом AES та створення й перевірка цифрових підписів за допомогою асиметричного алгоритму ECDSA.

Для розробки користувацького інтерфейсу чату було використано бібліотеку Swing з пакету `javax.swing.*`. Swing надає потужні та гнучкі компоненти для створення графічного інтерфейсу користувача. Створено вікно чату (`JFrame`), текстову область для відображення повідомлень

(JTextArea), смугу прокрутки (JScrollPane) та текстове поле для введення повідомлень (JTextField). Це забезпечує зручний та інтуїтивно зрозумілий інтерфейс користувача.

Для забезпечення мережної комунікації між клієнтами було використано класи Socket та ServerSocket з пакету java.net. Ці класи дозволяють встановлювати мережні з'єднання та обмінюватися даними між клієнтами.

Під час розробки проекту було використано середовище розробки IntelliJ IDEA 2024.1.1. Це потужна та зручна IDE, яка надає широкий набір інструментів для розробки на Java, включаючи інтелектуальне автодоповнення коду, налагодження та інтеграцію з системами контролю версій.

При виборі технологій та інструментів для розробки було взято до уваги такі фактори, як безпека та зручність використання. В розробленому застосунку забезпечено надійний захист конфіденційності повідомлень за допомогою сучасних криптографічних алгоритмів та протоколів, в поєднанні зі зрозумілим та зручним інтерфейсом користувача.

2.2 Вимоги до системи та функціональні специфікації

Основна мета створення системи - забезпечення конфіденційного та безпечного обміну текстовими повідомленнями між користувачами. Ключові вимоги включають реалізацію надійного шифрування повідомлень та автентифікацію користувачів. Функціональність зосереджена на обміні текстовими повідомленнями в режимі реального часу без передачі файлів, створення груп чи підтримки аудіо/відео дзвінків.

Для забезпечення безпеки використовується комбінація симетричного шифрування AES для зашифрування/розшифрування повідомлень та

асиметричного алгоритму цифрового підпису ECDSA для перевірки автентичності й цілісності. Автентифікація користувачів є обов'язковою і здійснюється за допомогою їх персональних сертифікатів.

Система розрахована на роботу в ОС Windows без інтеграції з іншими сервісами.

2.3 Архітектура захищеного P2P чату

Архітектура застосунку, що розробляється, представляє собою повністю децентралізовану P2P систему, де кожен вузол є одночасно і клієнтом, і сервером. Немає центрального сервера, вузли безпосередньо підключаються один до одного та обмінюються повідомленнями.

Клієнт (AliceClient):

- прослуховує вхідні підключення від інших клієнтів на заданому порту за допомогою класу `'ServerSocket'`.
- при отриманні нового підключення створює екземпляр класу `'ClientHandler'` для обробки взаємодії з кожним клієнтом в окремому потоці.
- зберігає список підключених клієнтів (`'writers'`) для розсилки їм повідомлень.
- підключається до інших клієнтів, вводячи їхні IP-адреси та порти.
- під час підключення відбувається обмін сертифікатами OpenSSL між клієнтами для автентифікації.

Обробник клієнта (ClientHandler):

- відповідає за взаємодію з окремим підключеним клієнтом через встановлене сокетне з'єднання.
- очікує вхідні повідомлення від клієнта, розшифровує їх за допомогою симетричного алгоритму AES і перевіряє цифровий підпис ECDSA, використовуючи відкритий ключ із сертифіката відправника.

- у разі успішної перевірки підпису відображає повідомлення в інтерфейсі користувача.

Криптографічні механізми:

А). Симетричне шифрування AES

1) використовується для шифрування та розшифрування повідомлень, якими обмінюються клієнти.

2) для кожної сесії обміну повідомленнями генерується новий випадковий ключ AES, який передається іншій стороні в зашифрованому вигляді.

3) перед надсиланням повідомлення клієнт шифрує його за допомогою сесійного ключа AES, використовуючи метод `'encryptMessage'` класу.

4) При отриманні зашифрованого повідомлення клієнт розшифровує його за допомогою сесійного ключа AES, використовуючи метод `'decryptMessage'` класу `'ClientHandler'`.

Б). Цифровий підпис ECDSA

1) забезпечує цілісність і автентичність повідомлень, якими обмінюються клієнти.

2) кожен клієнт генерує свою унікальну пару ключів (приватний і публічний) за допомогою методу `'generateKeyPair'` класу `'Peer'`.

3) перед надсиланням повідомлення клієнт підписує його своїм приватним ключем, використовуючи метод `'sign'` класу `'Signature'`.

4) цифровий підпис, зашифроване повідомлення і відкритий ключ клієнта надсилаються іншим підключеним клієнтам.

5) клієнти перевіряють цифровий підпис повідомлення за допомогою відкритого ключа відправника із сертифіката, використовуючи метод `'verify'` класу `'Signature'`.

Така децентралізована архітектура без сервера забезпечує високий рівень конфіденційності спілкування завдяки шифруванню повідомлень за допомогою AES, а також забезпечує цілісність і автентичність через

використання цифрових підписів ECDSA та обмін сертифікатами OpenSSL між клієнтами.

2.4 Моделювання системи з використанням UML діаграм

Діаграма варіантів використання (use case) для AliceClient ілюструє, як Аліса взаємодіє з клієнтським додатком до виконання різних завдань. Аліса відкриває вікно чату в програмі, щоб розпочати або продовжити спілкування. Вона вибирає цифровий сертифікат для автентифікації та шифрування повідомлень, що є важливим для забезпечення безпеки спілкування.

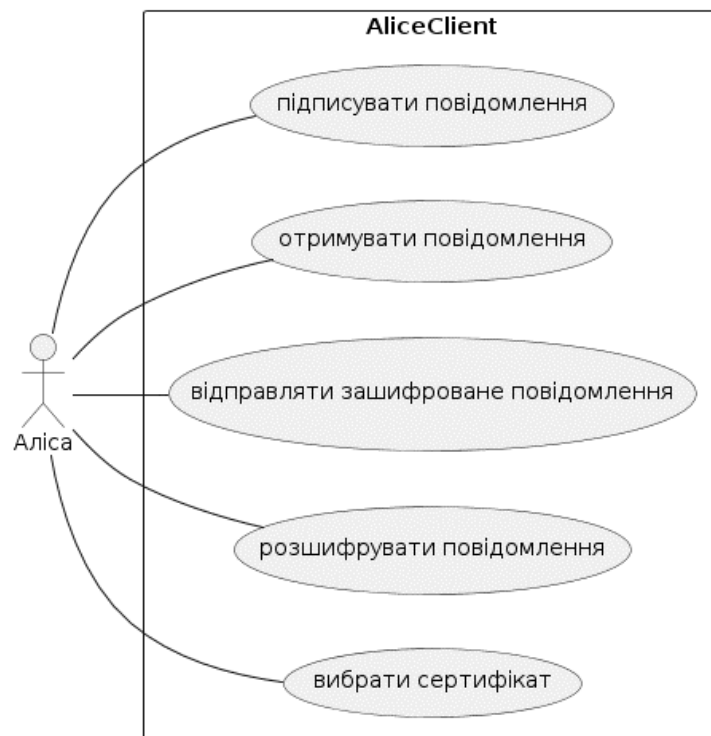


Рисунок 2.2 - Діаграма use case

Підключення до іншого користувача в децентралізованій системі включає автентифікацію та обмін сертифікатами безпосередньо між користувачами. Цей процес розширюється варіантом використання "Отримати Повідомлення", що означає, що отримання повідомлень є

частиною процесу підключення, і включає процес "Розшифрувати Повідомлення". Аліса також надсилає зашифроване повідомлення іншому користувачеві.

Діаграма показує основні функції клієнтської програми та те, як клієнт взаємодіє з ним для забезпечення безпечного обміну повідомленнями. Розшифровування та шифрування повідомлень, вибір сертифікатів та підключення до інших користувачів – всі ці дії допомагають створити захищене середовище для спілкування.

На діаграмі показаний процес управління захищеним з'єднанням та обміну зашифрованими повідомленнями між двома клієнтами – Алісою та Бобом у децентралізованій системі обміну повідомленнями.

Спочатку Аліса обирає свій цифровий сертифікат та надсилає інформацію про клієнта Боба. Цифровий сертифікат використовується для автентифікації клієнта та забезпечення безпеки з'єднань. Боб отримує цю інформацію та відображає дані про сертифікат Аліси.

Далі Боб ініціює підключення до Аліси. Після успішного під'єднання, Аліса надсилає зашифроване повідомлення Бобу. Повідомлення шифрується з використанням криптографічного алгоритму, такого як AES.

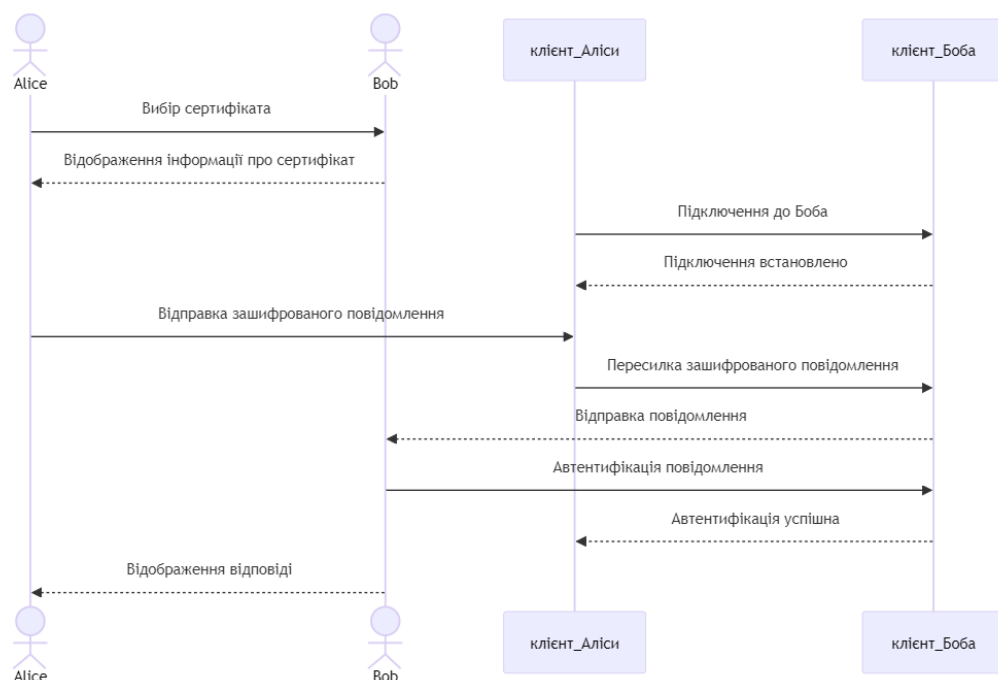


Рисунок 2.1 - Діаграма послідовності

Таким чином, діаграма алгоритму послідовностей забезпечує швидкий процес обміну даними для аутентифікації клієнтів, створює захищені сертифікатні з'єднання між ними з використанням шифрування, а також передає зашифровані повідомлення в обидва напрями для перевірки працездатності системи.

Після успішної автентифікації користувач може відправляти повідомлення. Кожне повідомлення, перед відправкою, підписується цифровим підписом з використанням приватного ключа відправника. Підписане повідомлення відправляється в зашифрованому виді до іншого клієнта, де відбувається перевірка цифрового підпису за допомогою публічного ключа відправника. Якщо підпис є дійсним, повідомлення вважається автентичним і обробляється далі. У разі невдалої перевірки підпису повідомлення відхиляється як потенційно скомпрометоване.

2.5 Графічний інтерфейс користувача та обробка взаємодії з користувачем

Створення вікна чату:

- використання класу `JFrame` для створення основного вікна програми.
- додавання областей для відображення історії чату (`JTextArea`) і введення повідомлень (`TextField`).
- налаштування розміру вікна та його поведінки при закритті.

Обробка введення повідомлень:

- додавання обробника події для `TextField`, який викликатиметься при натисканні клавіші `Enter`.

- отримання тексту введеного повідомлення і виконання необхідних дій (шифрування, підписання, розсилка).
 - відображення відправленого повідомлення в області історії чату.
- Відображення вхідних повідомлень:
- обробка вхідних повідомлень в окремому потоці (ClientHandler).
 - розшифровка і перевірка підпису отриманого повідомлення.
 - використання `SwingUtilities.invokeLater()` для безпечного оновлення GUI з потоку обробника.
 - відображення отриманого повідомлення в області історії чату із зазначенням відправника.

Обробка введення адрес інших клієнтів:

- використання `JOptionPane` для отримання IP-адреси клієнта для підключення.
- встановлення з'єднання з віддаленим клієнтом і додавання його до списку `writers` для розсилки повідомлень.

2.6 Висновки за результатами другого розділу

У другому розділі роботи проведено детальне моделювання системи та розробка інтерфейсу для захищеного P2P чату. З використанням UML діаграм, таких як діаграми класів, послідовностей та варіантів використання, була візуалізована структура системи та взаємодія її компонентів. Докладно були розглянуті елементи графічного інтерфейсу, включаючи вікно чату та області для відображення та введення повідомлень. Також описані методи обробки дій і взаємодій з інтерфейсом. Моделювання системи дозволило структурувати процес розробки і створити зрозумілу і логічно пов'язану архітектуру, що в кінцевому підсумку забезпечило реалізацію зручного і функціонального інтерфейсу для P2P чату.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЗАХИЩЕНОГО P2P ЧАТУ

3.1 Реалізація основних компонентів P2P чату

Прослуховування вхідних з'єднань:

- у класі `AliceClient` визначено константу `LISTEN_PORT`, яка вказує порт, на якому клієнт прослуховуватиме вхідні з'єднання від інших клієнтів (у даному випадку 12345).
- у методі `main` класу `AliceClient` створюється екземпляр `ServerSocket` із зазначеним портом `LISTEN_PORT`. `ServerSocket` прослуховує вхідні підключення на вказаному порту.
- клієнт входить у нескінченний цикл, очікуючи нових підключень від інших клієнтів за допомогою методу `accept()` класу `ServerSocket`.
- коли інший клієнт встановлює з'єднання, `accept()` повертає об'єкт `Socket`, що представляє встановлене з'єднання.
- для кожного підключеного клієнта створюється екземпляр класу `ClientHandler`, який опрацьовує взаємодію з цим клієнтом.
- кожен екземпляр `ClientHandler` запускається в окремому потоці за допомогою `new Thread(clientHandler).start()`. Це дозволяє клієнту обробляти кількох підключених клієнтів одночасно.

```
new Thread() -> {
    try (ServerSocket serverSocket = new ServerSocket(LISTEN_PORT)) {
        while (true) {
            Socket socket = serverSocket.accept();
            new Thread(new ClientHandler(socket, alice)).start();
        }
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}
```

Підключення до інших клієнтів:

- у класі `AliceClient` користувачеві пропонується ввести IP-адресу іншого клієнта для підключення.

- встановлюється з'єднання з іншим клієнтом за допомогою створення об'єкта `Socket`. У конструкторі `Socket` вказується IP-адреса іншого клієнта та порт `CONNECT_PORT`, на якому інший клієнт очікує підключення.
- після успішного підключення клієнт `AliceClient` отримує вихідний потік (`PrintWriter`) для надсилання даних іншому клієнту через встановлене сокетне з'єднання.
- клієнт `AliceClient` надсилає інформацію про свій сертифікат іншому клієнту.
- для кожного нового підключення створюється екземпляр класу `ClientHandler`, який запускається в окремому потоці для обробки взаємодії з цим клієнтом.

```
try {
    Socket socket = new Socket(remoteIP, CONNECT_PORT);
    alice.addWriter(new PrintWriter(socket.getOutputStream(), true));
    new Thread(new ClientHandler(socket, alice)).start();

    // Надсилання інформації про сертифікат
    alice.broadcast("CERT|" + certInfo);
}
catch (IOException e) {
    e.printStackTrace();
}
```

Обмін повідомленнями:

- коли клієнт надсилає повідомлення, воно шифрується за допомогою AES, підписується цифровим підписом та відправляється всім підключеним клієнтам через вихідний потік (`PrintWriter`).
- кожен підключений клієнт отримує повідомлення від клієнта через вхідний потік (`BufferedReader`) в класі `ClientHandler`.
- клієнт розшифровує отримане повідомлення, перевіряє цифровий підпис та відображає повідомлення у вікні чату.

```
(BufferedReader) та відображає їх у графічному інтерфейсі чату.
inputField.addActionListener(e -> {
    String message = inputField.getText();
    if (!message.isEmpty()) {
        try {
            // Генерація нового AES-ключа для кожної сесії
            SecretKey sessionKey = generateAESKey();
            String sessionKeyString = Base64.getEncoder().encodeToString(sessionKey.getEncoded());

            // Підпис повідомлення
            Signature signer = Signature.getInstance("SHA256withECDSA");
```

```

signer.initSign(alice.keyPair.getPrivate());
signer.update(message.getBytes());
byte[] signatureBytes = signer.sign();
String signature = Base64.getEncoder().encodeToString(signatureBytes);

// Шифрування повідомлення з використанням нового AES-ключа
String encryptedMessage = encryptMessage(message, sessionKey);

// Формування повного повідомлення із передачею зашифрованого AES-ключа
String fullMessage = "MSG|Alice|" + encryptedMessage + "|" + signature + "|" + sessionKeyString + "|" +
Base64.getEncoder().encodeToString(alice.getPublicKey().getEncoded());
alice.broadcast(fullMessage);
inputField.setText("");
chatArea.append("Me: " + message + "\n");
}
catch (Exception ex) {
    ex.printStackTrace();
}
}
});

```

Завершення підключення:

- коли клієнт закриває програму або виникає помилка, всі встановлені з'єднання з іншими клієнтами закриваються за допомогою методу `close()` об'єкта `Socket`.
- на стороні підключених клієнтів, коли клієнт відключається, відповідний екземпляр `ClientHandler` виявляє закриття з'єднання і видаляє асоційований екземпляр `PrintWriter` зі списку `writers` в об'єкті `Peer`.
- при видаленні `PrintWriter` з списку `writers`, клієнт більше не зможе надсилати повідомлення відключеному клієнту.
- клієнт може транслювати повідомлення про відключення клієнта решті підключених клієнтів через збережені екземпляри `PrintWriter` в списку `writers`.

```

public void run() {
    try (PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {
        peer.addWriter(out);
        String message;
        while ((message = in.readLine()) != null) {
            System.out.println("Received message: " + message); //для вивода
            String[] parts = message.split("\\|", 2);
            if (parts.length == 2) {
                String type = parts[0];
                String content = parts[1];
                if ("CERT".equals(type)) {
                    SwingUtilities.invokeLater(()->peer.chatArea.append("Received certificate info:\n" + content + "\n"));
                }
                else if ("MSG".equals(type)) {
                    String[] msgParts = content.split("\\|");

```

```

if (msgParts.length == 5) {
    String sender = msgParts[0];
    String encryptedMessage = msgParts[1];
    String signature = msgParts[2];
    String sessionKeyString = msgParts[3];
    String publicKeyString = msgParts[4];
    SecretKey sessionKey = new SecretKeySpec(Base64.getDecoder().decode(sessionKeyString), "AES");
    PublicKey publicKey = getPublicKeyFromString(publicKeyString);
    String decryptedMessage = decryptMessage(encryptedMessage, sessionKey);
    if (verifySignature(publicKey, decryptedMessage, signature)) {
        SwingUtilities.invokeLater(()->peer.chatArea.append(sender + ": " + decryptedMessage + "\n"));
    }
    else {
        System.out.println("Signature verification failed for message: " + encryptedMessage);
    } } } } }
catch (Exception e) {
    e.printStackTrace();
}
}

```

3.2 Інтеграція алгоритмів шифрування та аутентифікації

Для забезпечення надійного захисту конфіденційності, цілісності та автентичності повідомлень у розробленому P2P чаті було вирішено використати комбінацію симетричного шифрування AES та асиметричного алгоритму цифрового підпису ECDSA. Ця комбінація криптографічних механізмів дозволяє досягти високого рівня безпеки під час обміну повідомленнями між клієнтами безпосередньо.

Симетричне шифрування AES відіграє ключову роль у забезпеченні конфіденційності повідомлень. Кожне повідомлення шифрується перед відправкою за допомогою ключа AES, який є спільним секретом між клієнтами. Це означає, що навіть якщо зловмисник перехопить повідомлення, він не зможе розшифрувати його вміст без знання ключа AES. Таким чином, конфіденційність повідомлень залишається захищеною протягом усього процесу передачі.

З іншого боку, цифровий підпис ECDSA використовується для забезпечення цілісності та автентичності повідомлень. Кожен клієнт має свою унікальну пару ключів – приватний ключ, який використовується для

створення цифрового підпису, та публічний ключ, який використовується для перевірки підпису.

Перед надсиланням повідомлення клієнт створює цифровий підпис, використовуючи свій приватний ключ та саме повідомлення. Цей підпис автоматично перевіряється на стороні клієнтів. Перевірка дозволяє переконатися, що повідомлення було надіслано вказаним клієнтом і не було змінено під час передачі. Користувачеві не потрібно вручну перевіряти цифровий підпис, ця процедура виконується програмою автоматично при отриманні повідомлення.

Таким чином, інтеграція цифрового підпису ECDSA забезпечує автоматичну перевірку цілісності та автентичності повідомлень у децентралізованій системі P2P обміну повідомленнями без необхідності ручної перевірки користувачем.

Автентифікація:

- під час підключення клієнта відбувається процес автентифікації.
- клієнт очікує у користувача його підписаний сертифікат, який був згенерований через OpenSSL.
- клієнт відправляє інформацію, яка знаходиться у цьому сертифікаті, всім підключеним користувачам до чату.

```
private static X509Certificate chooseCertificate() {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle("Виберіть сертифікат OpenSSL");
    int result = fileChooser.showOpenDialog(null);
    if (result == JFileChooser.APPROVE_OPTION) {
        File selectedFile = fileChooser.getSelectedFile();
        try (FileInputStream fis = new FileInputStream(selectedFile)) {
            CertificateFactory cf = CertificateFactory.getInstance("X.509");
            return (X509Certificate)cf.generateCertificate(fis);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
    return null;
}
```

Шифрування повідомлень:

- перед надсиланням повідомлення клієнт шифрує його за допомогою алгоритму AES та загального ключа.
- зашифроване повідомлення кодується у формат Base64 перед надсиланням через мережу.
- при отриманні зашифрованого повідомлення клієнт використовує метод `decryptMessage`, який розшифровує повідомлення за допомогою того ж ключа AES.

```
private static SecretKey generateAESKey() throws NoSuchAlgorithmException {
    KeyGenerator keyGen = KeyGenerator.getInstance("AES");
    keyGen.init(256); // Можна використовувати 128, 192 або 256 бітні ключі
    return keyGen.generateKey();
}

private static String encryptMessage(String message, SecretKey key) throws Exception {
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    byte[] encryptedBytes = cipher.doFinal(message.getBytes());
    return Base64.getEncoder().encodeToString(encryptedBytes);
}

private String decryptMessage(String encryptedMessage, SecretKey key) throws Exception {
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.DECRYPT_MODE, key);
    byte[] encryptedBytes = Base64.getDecoder().decode(encryptedMessage);
    byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
    return new String(decryptedBytes);
}
```

Цифровий підпис:

- клієнт генерує пару ключів (публічний та приватний) з використанням алгоритму ECDSA.
- перед надсиланням повідомлення клієнт створює цифровий підпис, використовуючи свій приватний ключ та алгоритм "SHA256withECDSA".

```
// Підпис повідомлення
Signature signer = Signature.getInstance("SHA256withECDSA");
signer.initSign(alice.keyPair.getPrivate());
signer.update(message.getBytes());
byte[] signatureBytes = signer.sign();
String signature = Base64.getEncoder().encodeToString(signatureBytes);
```

- підпис кодується у формат Base64 і надсилається разом із повідомленням.
- при отриманні повідомлення, клієнт отримує публічний ключ з отриманого рядка за допомогою методу `getPublicKeyFromString`.

```
private KeyPair generateKeyPair() throws NoSuchAlgorithmException, InvalidAlgorithmParameterException{
    KeyPairGenerator keyGen = KeyPairGenerator.getInstance("EC");
    keyGen.initialize(new ECGenParameterSpec("secp256r1"));
    return keyGen.generateKeyPair();
}
```

- клієнт перевіряє підпис за допомогою методу `verifySignature`, використовуючи публічний ключ клієнта.

```
private boolean verifySignature(PublicKey publicKey, String message, String signatureStr) throws Exception {
    Signature signature = Signature.getInstance("SHA256withECDSA");
    signature.initVerify(publicKey);
    signature.update(message.getBytes());
    byte[] signatureBytes = Base64.getDecoder().decode(signatureStr);
    return signature.verify(signatureBytes);
}
```

Обмін повідомленнями:

- клієнт надсилає повідомлення до іншого клієнта, у форматі "ім'я відправника, зашифроване повідомлення, підпис, публічний ключ".

```
// Формування повного повідомлення із передачею зашифрованого AES-ключа
String fullMessage = "MSG|Alice|" + encryptedMessage + "|" + signature + "|" + sessionKeyString + "|" +
Base64.getEncoder().encodeToString(alice.getPublicKey().getEncoded());
alice.broadcast(fullMessage);
inputField.setText("");
chatArea.append("Me: " + message + "\n");
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

- клієнт розшифровує повідомлення, перевіряє підпис та транслює повідомлення всім підключеним клієнтам.

- клієнт відображає отримані повідомлення у графічному інтерфейсі чату.

3.3 Тестування функціональності та безпеки P2P чату

На цьому скріншоті представлено вікно вибору сертифіката користувача. Це важливий крок для безпеки, оскільки сертифікат служить засобом автентифікації користувача в системі.

Вибір сертифіката гарантує, що лише авторизовані користувачі зможуть підключитися до мережі P2P чату та обмінюватися зашифрованими повідомленнями.

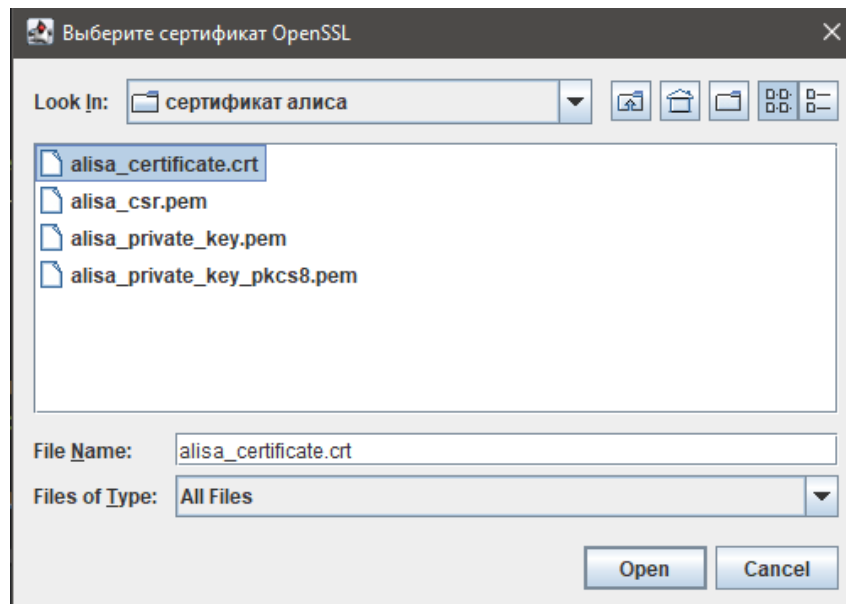


Рисунок 3.1 - Вибір сертифіката користувача

Сертифікати також відіграють ключову роль у процесі шифрування та розшифрування повідомлень.

Цей скріншот показує інтерфейс для введення IP-адреси іншого користувача. IP-адреса необхідна для встановлення прямого з'єднання між користувачами в мережі P2P.

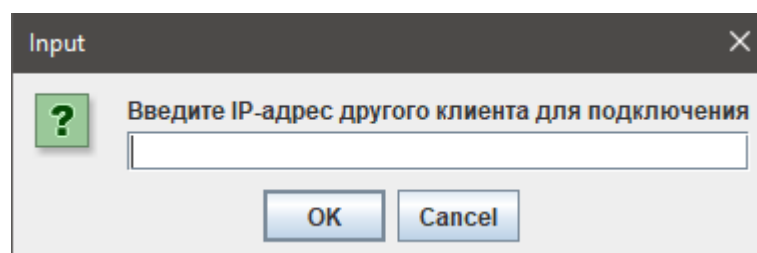


Рисунок 3.2 - Введення IP-адреси

Правильне введення IP-адреси дозволяє надсилати та отримувати повідомлення безпосередньо, минаючи центральні сервери, що підвищує конфіденційність та стійкість системи до атак.

На скріншоті показаний чат між користувачами на ім'я Боб і Аліса. Тут можна спостерігати, як відбувається обмін повідомленнями у реальному часі.

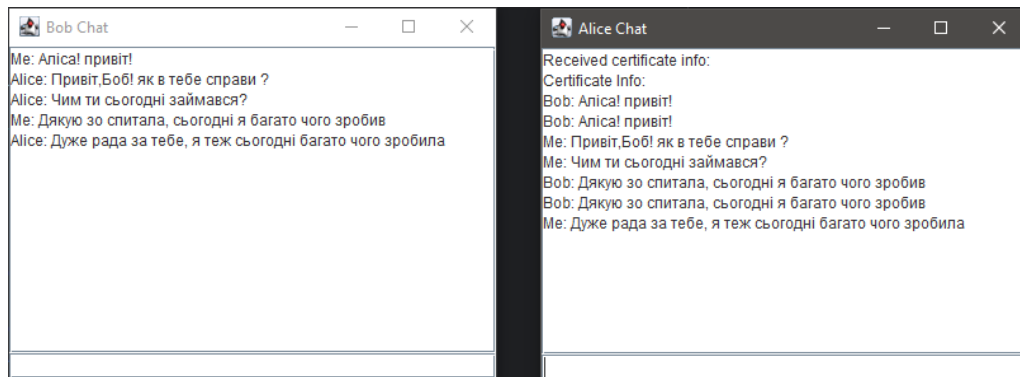


Рисунок 3.3 - Чат Боба та Аліси

Надіслані повідомлення шифруються, а отримані - розшифровуються, що забезпечує високий рівень безпеки та конфіденційності комунікації. Цей процес підтверджує, що система працює коректно та безпечно.

Після того як інший клієнт під'єднався, інформація про його сертифікат відправляється іншому клієнту, це зроблено для того, щоб інший користувач, знав хто під'єднався. Інформація з сертифікату може мати таку інформацію як ім'я користувача, термін дії сертифіката, організацію, що видає, та інші важливі деталі.

```
Received message: CERT|Certificate Info:
Received message: Subject: EMAILADDRESS=bob.com, CN=ZNTU, OU=ZNTU, O=ZNTU, L=ZP, ST=Bob RT-710, C=UA
Received message: Issuer: EMAILADDRESS=bob.com, CN=ZNTU, OU=ZNTU, O=ZNTU, L=ZP, ST=Bob RT-710, C=UA
Received message: Valid From: Tue May 28 15:38:54 MSK 2024
Received message: Valid To: Wed May 28 15:38:54 MSK 2025
```

Рисунок 3.4 - Інформація про сертифікат

Перегляд інформації про сертифікат допомагає переконатися в справжності та актуальності сертифіката, що необхідно для запобігання підrobкам та несанкціонованому доступу.

На цьому скріншоті видно, як клієнтська програма отримала повідомлення від користувача на ім'я Боб, а також відображає зашифроване повідомлення від користувача на ім'я Аліса.

```
Received message: MSG|Bob|AI3GqgsKvk+gZJ2fITrZIWf5v0VV7prJ3E6ps/
Broadcasting message: MSG|Alice|THCdZm+gc8i4Sf/v//WakD16sZasXqti
```

Рисунок 3.5 - Отримане повідомлення від Боба та зашифроване повідомлення від Аліси

Це підтверджує, що повідомлення успішно передаються і шифруються між користувачами, гарантуючи безпеку даних, що передаються.

Тут представлено отримання повідомлення від Аліси та зашифроване повідомлення від Боба. Цей скріншот аналогічний до попереднього і демонструє зворотний процес обміну повідомленнями.

```
Broadcasting message: MSG|Bob|AI3GqgsKvk+gZJ2fITrZIWf5v0VV7prJ3E
Received message: MSG|Alice|THCdZm+gc8i4Sf/v//WakD16sZasXqtiEC0m
```

Рисунок 3.6 - Отримане повідомлення від Аліси та зашифроване повідомлення від Боба

Це підтверджує двосторонню працездатність системи, де обидва користувачі можуть надсилати та отримувати зашифровані повідомлення.

На цьому скріншоті показано помилку верифікації цифрового підпису. Це важливий елемент тестування безпеки, оскільки показує, що система здатна виявляти та відхиляти повідомлення з некоректними чи підробленими підписами.

```
C:\Users\dom\.jdk\liberica-1.8.0_412\bin\java.exe ...
Failed to verify signature
Failed to verify signature
```

Рисунок 3.7 - Помилка верифікації підпису

Це запобігає можливості надсилання підроблених повідомлень і забезпечує цілісність даних, що передаються.

ВИСНОВКИ

В дипломній роботі проаналізовано архітектуру P2P мереж, протоколи прямої взаємодії між клієнтами та методи шифрування повідомлень та запропоновано програмне рішення захищеного P2P-чату для безпечного обміну повідомленнями в одноранговій мережі.

Для вирішення поставленої задачі використано комбінацію симетричного шифрування AES та асиметричного алгоритму цифрового підпису ECDSA, що забезпечує високий рівень безпеки, конфіденційність, цілісність та автентичність повідомлень. Особлива увага приділена захисту від таких атак, як "людина посередині", атака Сібілі, аналізу трафіку та розкриття IP-адрес.

В ході дипломній роботі розроблено проект програмного застосунку з використанням уніфікованої мови моделювання UML та програмний код захищеного P2P-чату з інтегрованими функціями шифрування та автентифікації повідомлень на мові програмування java з підключенням відповідних бібліотек. Розроблено зручний користувацький інтерфейс. Виконано тестування розробленого застосунку.

Практична значущість роботи заключається в можливості використання розробленого захищеного P2P-чату для безпечного обміну автентифікованими повідомленнями в одноранговій мережі з недовірою сторін.

ПЕРЕЛІК ВИКОРИСТАНИХ ПОСИЛАННЯ

1. P2P-network.svg. Вікіпедія. URL: <http://surl.li/ulfha>
2. Що таке NAT та для чого використовують цей стандарт. URL: <https://hyperhost.ua/info/ru/chto-takoe-nat-dlya-chego-ispolzuyu-dannyu-standart>
3. Горбенко І. Д., Горбенко Ю. І. Прикладна криптологія: теорія, практика, застосування: монографія. Харків: Форт, 2012. 868 с.
4. Що таке WebRTC та чого розробники та компанії люблять цю мережеву технологію. URL: <https://senior.ua/articles/chto-takoe-webrtc-i-rochemu-razrabotchiki-i-kompanii-lyubyat-etu-setevuyu-tehnologiyu>
5. How it works. Briar. URL: <https://briarproject.org/how-it-works/>
6. Гриненко Т. О., Клименко Л. А., Ковальчук Л. В. Дослідження методів криптографічного захисту інформації. Сучасний захист інформації. 2016. № 4. С. 68-70.
7. Асиметричні алгоритми шифрування. URL: <http://surl.li/udiqi>
8. Остапов С. Е., Евсеев С. П., Король О. Г. Технології захисту інформації: навчальний посібник. Харків: Вид. ХНЕУ, 2013. 476 с.
9. Атаки на криптовалюту: види та методи захисту. Bitfury. URL: <https://habr.com/ru/companies/bitfury/articles/340378/>
10. Коваленко Є. Г., Кучук Г. А., Рубан І. В. Методи та моделі безпеки розподілених інформаційних систем: монографія. Харків: Друкарня Мадрид, 2018. 214 с.
11. Петров В.П., Петров С.В. Информационная безопасность человека и общества. М.: ЭНАС, 2007. 334 с
12. Дудикевич В. Б., Опіський І. Р., Гарасим Ю. Р. Основи забезпечення конфіденційності та цілісності інформації в розподілених системах. Вісник Національного університету "Львівська політехніка". Автоматика, вимірювання та керування. 2012. № 741. С. 160.

13. What is a man-in-the-middle attack and how to prevent it. Panda Security. URL: <https://www.pandasecurity.com/en/mediacenter/man-in-the-middle-attack/>
14. Sybil attack scenario in a P2P network. ResearchGate. URL: https://www.researchgate.net/figure/Sybil-attack-scenario-in-a-P2P-network_fig1_369347498
15. Кузнецов О. О., Євсєєв С. П., Король О. Г. Захист інформації в інформаційних системах: навчальний посібник. Харків: Вид. ХНЕУ, 2011. 510 с.
16. Соціальна інженерія / Inform безпека. URL: <https://news.koblevo.org.ua/sotsyalnaya-ynzheneryya-inform-bezopasnost/>
17. Фішинг: як йому протистояти? Root Nation. URL: <https://root-nation.com/ru/posts/internet-posts/ru-phishing-kak-emu-protivostoyat/>