Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФКАЦІЙНА РОБОТА

на тему: «Інформаційна технологія захищеного обміну даними на основі пірингових мереж»

08-20.МКР.007.00.000 ПЗ

Виконав: студент 2 курсу, групи 1БС-21м
спеціальності 125 Кібербезпека
_____ Притула А. В.

Керівник: к. т. н., доц. каф ЗІ
_____ Куперштейн Л. М.
«19» грудня 2022 р.

Опонент: к. т. н., доц. каф ПЗ
_____ Хошаба О. М.
«19» грудня 2022 р.

Допущено до захисту

Завідувач кафедри ЗІ, д.т.н., проф
_____ Лужецький В. А.
«19» грудня 2022 р.

Вінниця – 2022 р.

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Зав. кафедри ЗІ, д.т.н, проф.
_____В. А. Лужецький
«_15_» _вересня_ 2022 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Притулі Андрію Вікторовичу

1. Тема роботи: «Інформаційна технологія захищеного обміну даними на основі піринговых мереж»
керівник роботи: Куперштейн Леонід Михайлович, к. т. н., доц. каф. ЗІ, затверджені наказом ректора ВНТУ від 14 вересня 2022 року №203.

2. Строк подання студентом роботи: 19 грудня 2022 р.

3. Вихідні дані до роботи:
– операційна система – підтримує кросплатформність;
– технологія передачі даних – peer-to-peer;
– середовище розробки – Intellij IDEA;
– мова програмування – Dart;
– фреймворк – Flutter.

4. Зміст текстової частини: Вступ. 1. Аналіз предметної області. 2. Розробка інформаційної технології. 3. Програмна реалізація інформаційної технології. 4. Економічна частина. Висновки. Список використаних джерел. Додатки.

5. Перелік ілюстративного матеріалу: Складові інформаційної технології (плакат, А4). Схема процесу автентифікації (плакат, А4). Схема процесу створення ключів шифрування (плакат, А4). Схема процесу створення унікального ідентифікатора (плакат, А4). Схема процесу встановлення з'єднання (плакат, А4). Схема процесу передачі даних (плакат, А4). Загальна архітектура системи (плакат, А4). Загальний алгоритм роботи додатку (плакат, А4). Загальний алгоритм роботу модулю автентифікації (плакат, А4). Загальний алгоритм генерування ключів шифрування (плакат, А4). Загальна схема обміну ключами

шифрування (плакат, А4). Схема передачі даних по мережі (плакат, А4).

Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Куперштейн Л. М., к.т.н., доц. кафедри ЗІ		
2	Куперштейн Л. М., к.т.н., доц. кафедри ЗІ		
3	Куперштейн Л. М., к.т.н., доц. кафедри ЗІ		
4	Лесько Олександр Йосипович к.е.н., доц, професор кафедри ЕПВМ		

6. Дата видачі завдання 1 вересня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Зміст етапу	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	01.09.2022 – 04.09.2022	
2	Аналіз інформаційних джерел за напрямком магістерської кваліфікаційної роботи	05.09.2022 – 15.09.2022	
3	Науково-технічне обґрунтування	16.09.2022 – 22.09.2022	
4	Розробка технічного завдання	23.09.2022 – 04.10.2022	
5	Аналіз мереж та протоколів peer-to-peer	05.10.2022 – 08.10.2022	
6	Аналіз та формування вимог до програмного засобу	09.10.2022 – 16.10.2022	
7	Розробка програмного засобу для передавання інформації на основі піринговых мереж	17.10.2022 – 14.11.2022	
8	Тестування розробленого програмного засобу	15.11.2022 – 17.11.2022	
9	Розробка розділу економічного обґрунтування доцільності розробки	18.11.2022 – 21.11.2022	
10	Аналіз виконання ТЗ, висновки	22.11.2022 – 24.11.2022	
11	Оформлення пояснювальної записки	25.11.2022 – 29.11.2022	
12	Перевірка магістерської роботи на наявність плагіату	30.11.2022 – 02.12.2022	
13	Попередній захист та доопрацювання МКР	07.12.2022 – 19.12.2022	
14	Представлення МКР до захисту, рецензування	20.12.2022 – 21.12.2022	
16	Захист МКР	22.12.2022 – 26.12.2022	

Студент _____ (підпис) Притула А. В.

Керівник роботи _____ (підпис) Куперштейн Л. М.

УДК 004.

Притула

основі піринго

125 – Кібербезп

систем. Вінниц

На укр. м

Магістерс

технології захи

роботи було п

стандартну кліє

реалізації пірин

мережі. На осн

інформаційну т

а також алгорит

Ілюстрати

результатів розр

В еконо

інформаційної т

Ключові с

АНОТАЦІЯ

УДК 004.056

Притула А. В. Інформаційна технологія захищеного обміну даними на основі пірингових мереж. Магістерська кваліфікаційна робота зі спеціальності 125 – Кібербезпека, освітня програма – Безпека інформаційних і комунікаційних систем. Вінниця: ВНТУ, 2022. 103 с.

На укр. мові. Бібліогр.: 52 назв; рис.: 40 табл. 16.

Магістерська кваліфікаційна робота присвячена розробці інформаційної технології захищеного обміну даними на основі пірингових мереж. В рамках цієї роботи було проведено аналіз існуючих мереж передачі даних, порівняно стандартну клієнт-серверну архітектуру та пірингову, проаналізовано існуючі реалізації пірингових мереж, а також проаналізовані відомі атаки на пірингові мережі. На основі проаналізованих даних було запропоновано удосконалену інформаційну технологію для обміну даними. Розроблено структуру технології, а також алгоритми роботи.

Ілюстративна частина складається з 12 плакатів з демонстрацією результатів розробки і проведених досліджень

В економічному розділі здійснено оцінку витрат на розробку інформаційної технології.

Ключові слова: інформаційна безпека, пірингова мережа, P2P.

ABSTRACT

Prytula A. V. Information technology of secure data exchange based on peering networks. Master's thesis in the specialty 125 – cybersecurity, educational program - Security of information and communication systems. Vinnytsia: VNTU, 2022. – 103 p.

In Ukrainian. Bibliographer: 52 titles; fig .: 40 tabl. 16

The master's thesis is devoted to the development of information technology of secure data exchange based on peering networks. As part of this work, an analysis of existing data transmission networks was carried out, a standard client-server architecture and a peering architecture were compared, existing implementations of peering networks were analyzed, and known attacks on peering networks were analyzed. Based on the analyzed data, an improved information technology for data exchange was proposed. The technology structure and work algorithms have been developed.

The illustrative part consists of 12 posters with a demonstration of the results of development and conducted research

In the economic section, an assessment of costs for the development of information technology was made.

Keywords: information security, peering network, P2P.

ЗМІСТ

ВСТУП

На сьогоднішній день у світі не залишилось людей, які б не чули про Інтернет, комп'ютер, смартфон. Висока популярність інтернету пов'язана з тим, що дозволяє отримати потрібну інформацію будь-де і у будь-який час доби, але найголовніше, що він дозволяє людям спілкуватись незалежно від їх географічного розташування.

Оскільки дуже багато людей користуються веб-сайтами та месенджерами для спілкування, їх потрібно захищати, адже компрометація таких систем може нанести дуже велику шкоду користувачам, а особливо їх персональним даним. Особливо це відноситься до IT сектору. Під час розробки програмного забезпечення необхідно спілкуватись менеджерам, розробникам та замовникам між собою. Така комунікація вкрай важлива, адже розробка програмного забезпечення це довгий процес, який містить в собі велику кількість деталей без який програмне забезпечення буде поганої якості.

І з таким зростом користувачів інформація, яка передається по мережах набуває все більшої ціни. Це може бути і важлива комерційна інформація і інформація без секрету або персональна інформація, яка підлягає захисту. Для того, щоб запобігти втраті конфіденційності інформації потрібно використовувати сучасні методи обробки, передавання та зберігання інформації, а також мінімізувати час знаходження інформації в мережі.

Сучасні програми для спілкування зазвичай використовують клієнт-серверну архітектуру для побудови. Це означає, що всі дані, які надсилаються користувачами, зберігаються на віддаленому сервері. Такий спосіб збереження даних є небезпечним, оскільки зловмисник може скомпрометувати сервер і всі персональні дані користувачів можуть дістатись зловмиснику. Якщо ж говорити про розробку програмного забезпечення, то витік секретної інформації про алгоритми, які використовуються у програмах може повністю знищити не тільки саму програмку, а й цілу компанію, оскільки конкуренти можуть використати цю інформацію для покращення своїх застосунків.

На відміну від стандартних клієнт-серверних архітектур, пірингові не зберігають інформацію на центральному сервері, а тільки на пристроях користувачів. Такий спосіб зберігання конфіденційної інформації є більш стійким, адже для того, щоб отримати таку інформацію, потрібно напряму комунікувати з тим, в кого інформація є, наприклад методи соціальної інженерії або крадіжка пристрою. Але навіть якщо зловмисник отримав пристрій користувача, це не означає що він отримає інформацію, оскільки вона може бути зашифрована сучасними алгоритмами, для взлому яких потрібно дуже великі потужності та велика кількість часу. Тому на сьогодні, є досить актуальним використання піринговых мереж для передавання даних, адже інформація зберігається тільки на пристрої користувача. За допомогою піринговых мереж можливо розробити програмне забезпечення, яке є набагато стійкішим проти порушення конфіденційності, цілісності та доступності інформації на відміну від стандартних клієнт серверних мереж.

Об’єктом дослідження є процес захищеного обміну даними.

Предметом є методи та засоби захищеного обміну даними.

Метою магістерської кваліфікаційної роботи є підвищення захищеності процесів обміну даними за рахунок використання піринговых мереж.

Для досягнення поставленої мети необхідно розв’язати наступні задачі:

– проаналізувати мережі передачі даних;

– проаналізувати існуючі пірингові мережі;

– проаналізувати відомі атаки на пірингові мережі;

– розробити модель інформаційної технології;

– розробити алгоритми функціонування інформаційної технології;

– обрати засоби реалізації програмного засобу;

– визначити економічну доцільність від розробки засобу для захищеного передавання даних на основі піринговых мереж;

– реалізувати програмний засіб на основі розроблених моделей, алгоритмів та обраних засобів.

Наукова новизна полягаяє у тому, що вдосконалено інформаційну технологію захищеного обміну даними, яка полягає у використанні протоколу Діффі-Геллмана на основі еліптичних кривих і симетричного шифрування, що відрізняється використанням локальної бездротової піринговой архітектури мережі.

Практична цінність полягає у тому, що розроблено застосунок, який в повній мірі реалізує розроблені алгоритми передачі даних, шифрування, збереження даних. Розроблений застосунок є універсальним і може бути використаний будь-ким, наприклад маленькими та середніми IT компаніями або ж людьми, які знаходяться в одному товаристві, наприклад між мешканцями одного будинку для передавання даних, спілкування, в умовах обмеженого доступу до глобальної мережі.

Результати магістерської кваліфікаційної роботи доповідались на молодіжній науково-практичній інтернет-конференції студентів аспірантів та молодих науковців «КУСС-2022» (15-17 листопада 2022 р., м. Вінниця).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз мережевих архітектур

Передавання інформації відіграє важливу роль у житті кожного. Від передавання звичайного тексту до картинок і відео. Саме для виконання такої задачі було створено комп’ютерні мережі. Комп’ютери прийнято ділити на 2 категорії, клієнти і сервери. Клієнти – це звичайні користувачі послуг. Сервери – високопродуктивні машини, котрі мають виконувати запити клієнтів. Існує дві основні моделі, за допомогою яких дані передаються по мережі Peer-to-Peer (P2P) та клієнт-сервер.

Клієнт-серверна модель (багаторангова) – це централізована структура, яка розподіляє завдання або навантаження між постачальниками ресурсів або послуг, які називаються серверами, та запитувачами послуг, які називаються клієнтами [1]. Часто клієнти та сервери взаємодіють через комп'ютерну мережу на окремому обладнанні, але і клієнт, і сервер можуть перебувати в одній системі. Сервер починає виконувати свої програми, які спільно використовують свої ресурси з клієнтами. Зазвичай сервер не може користуватися інформацією клієнта без підтвердження або запиту зі сторони самого клієнта, при чому клієнт запитує інформацію або послуги з сервера. Таким чином, клієнти ініціюють сеанси зв'язку з серверами, які очікують вхідних запитів. Прикладами комп'ютерних програм, що використовують модель клієнт-сервер, є електронна пошта, мережевий друк та всесвітня павутина.

У цій мережевій моделі центральний сервер є обов'язковим, і всі клієнти підключаються до центрального сервера для отримання даних або використання його служб. На рис. 1.1 показано основні елементи, а саме:

- клієнти, тобто споживачі послуг;
- мережа, середовище передачі даних;
- сервер, постачальник послуг.

Сервер виступає у ролі кінцевої точки, очікуючи запити клієнтів, які він буде обробляти.

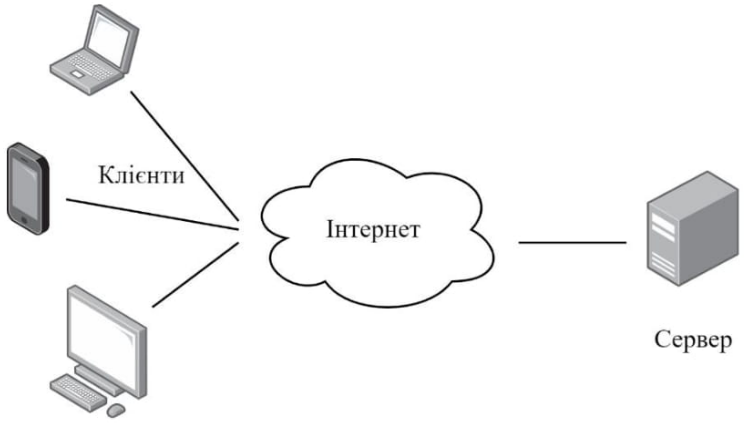


Рисунок 1.1 – Архітектура клієнт-серверної мережі

P2P модель (однорангова) – це модель розподілених мереж, яка розподіляє завдання або навантаження між вузлами. Однорангові вузли є однаково привілейованими, рівноправними учасниками мережі (рис. 1.2).

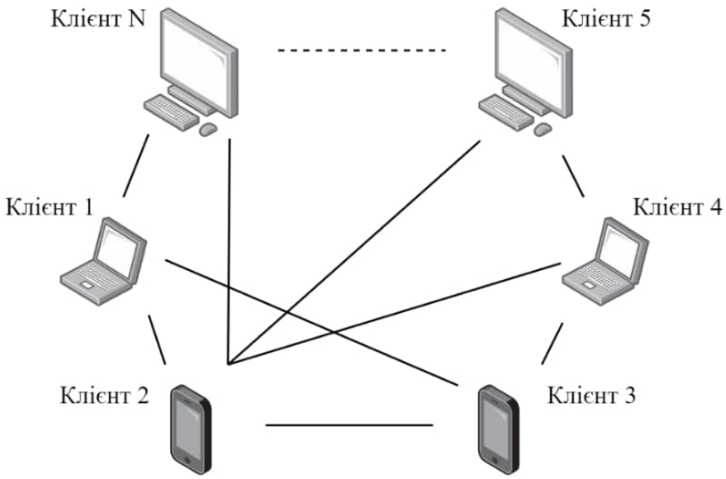


Рисунок 1.2 – Архітектура P2P мережі

Як наслідок, вони утворюють однорангову мережу вузлів [2]. Однорангові мережі надають частину своїх ресурсів, таких як обчислювальна потужність, дискове сховище або пропускна здатність мережі, безпосередньо іншим учасникам, без необхідності централізованої координації з боку серверів або стабільних вузлів [3]. Вузли є як постачальниками, так і споживачами ресурсів, на відміну від традиційної клієнт-серверної моделі, в якій споживання та

постачання ресурсів розділені [3]. Хоча системи P2P раніше використовувались у багатьох прикладних областях [4], архітектура була популяризована системою обміну файлами Napster, випущеною в 1999 році [5].

У таблиці 1.1 наведено порівняння P2P архітектури та клієнт-серверної.

Таблиця 1.1 – Порівняння клієнт-серверної та P2P архітектур

Характеристика	Клієнт-серверна архітектура	P2P архітектура
Передача даних	У клієнт-серверних архітектура сервер надає всі послуги та дані, тоді як клієнти запитують послуги та дані	У P2P-архітектурі всі вузли мережі діють як постачальники послуг і споживачі
Вартість	Архітектура клієнт-сервер дорого реалізувати, тому що центральний сервер має постійно працювати, інакше мережа вийде з ладу	P2P дешевше реалізовувати, ніж клієнт-серверну мережу, оскільки центральний сервер не потрібен
Розподіл пропускної здатності	Пропускна здатність залежить від швидкості підключення сервера до решти мережі	Повна пропускна здатність не виділяється заздалегідь у P2P з’єднанні. Він використовує вузол пропускної здатності відповідно до доступної пропускної здатності кожного вузла
Основний фокус	Обмін даними	Зв’язок та підключення
Зберігання даних	Дані зберігаються на централізованому сервері	Кожен вузол зберігає свої дані
Відмовостійкість	При збої на сервері мережа не може функціонувати	При збої одного вузла мережа продовжує функціонувати
Сервер	Сервер може бути перевантажений, коли багато клієнтів роблять одночасні запити на обслуговування	Сервер не є вузьким місцем, оскільки служби розподілені між численними серверами за допомогою однорангової мережі

Порівнюючи клієнт-серверну архітектура і P2P-архітектуру, P2P має наступні переваги:

– зменшення вартості. Системи, що мають центральний сервер і які працюють з багатьма клієнтами, потребують дуже великих коштів на підтримку. Але пірингова система намагається розподілити, навантаження між учасниками мережі;

– поєднання потенціалу. Піринговий підхід може використовуватись для того, щоб об’єднувати ресурси. Учасники мережі мають свої ресурси, як наприклад обчислювальна потужність або пам'ять. У разі, якщо якісь дії для виконання мають використовувати велику кількість ресурсів, як наприклад навчання нейронної мережі або створення моделей з великою кількістю вхідних параметрів, можна впровадити P2P, щоб розподілити виконання таких операцій між учасниками мережі;

– відмовостійкість. Мережа P2P є більш стабільною через свою структуру. Доки існує хоча б два вузли, мережа є активною. У клієнт-серверній архітектурі, відмова центрального вузла – відмова всієї мережі;

– збільшена автономія. Учасники пірингової системи не хочуть залежати від будь-якого центрального сервера або централізованого постачальника послуг. У архітектурі P2P, усі ресурси розподіляються між клієнтами, тобто є мало клієнтів, на кшталт сервера, у яких є вся інформація з інших клієнтів. Навіть у разі якщо такий клієнт знайдеться, то при його відключенні від мережі, інші клієнти зможуть дістати цю інформацію;

– збільшена конфіденційність. Користувачі, які потребують захисту приватної інформації є більш захищеними у такому типу мережі. При використанні криптографічних заходів, інформація, що передається по мережі має підвищений ступінь захисти, також це запобігає втручанню третьої сторони в процес передачі даних. На відміну від традиційних клієнт-серверних архітектур, інформація зберігається розподіллено, що змушує зловмисників витрачати час на пошук потрібної їм інформації;

Сьогодні P2P мережі настільки розвинулися, що world wide web вже не є найбільшою інформаційною мережею по ресурсах і породжуваному трафіку. Відомо, що кількість трафіку, обсяг переданих ресурсів в байтах, кількість хостів в пірингових, якщо їх розглядати в разом, нічим не поступаються мережі WWW. Більш того, трафік пірингових мереж становить 70% всього Інтернет-трафіку [6].

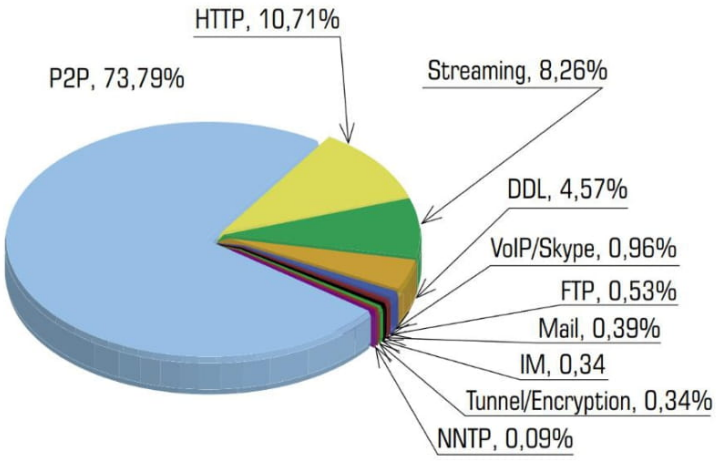


Рисунок 1.3 – Графік розподілу трафіку у мережі Інтернет

Існує багато областей застосування P2P мереж, що пояснюють їх зростаючу популярність, наприклад:

– обмін файлами. P2P виступають альтернативою FTP, оскільки вони втрачають популярність, зважаючи на значні інформаційні перевантаження;

– розподілені обчислення. Наприклад, такий P2P-проєкт, як SETI@home продемонстрував величезний обчислювальний потенціал для розпаралелених завдань [7];

– обмін повідомленнями. Раніше популярний месенджер ICQ – це P2P-проєкт;

– інтернет-телефонія, наприклад Skype;

– групова робота;

– кешування даних;

– блокчейн;

– розподілені фінанси;

– система електронної взаємодії державних електронних інформаційних ресурсів;

– резервне копіювання.

Загальновідомо, що система доменних імен в мережі Інтернет також фактично є мережею обміну даними, побудованою за принципом P2P.

Раніше популярна технологія GRID [8] – це система розподілених обчислень, яка використовувала технологію P2P для розподілення навантаження між користувачами. Ще прикладом може бути проєкт distributed.net, який використовує архітектуру P2P для розподілення навантаження для легального зламу криптографічних алгоритмів щоб перевірити їх надійність [9].

1.2 Аналіз пірингових мереж

На сьогоднішній день існує багато пірінгових мереж, а також протоколів для обміну даними за допомогою P2P, деякі з них вже закриті, але однозначно їх не варто забувати, адже за допомогою таких мереж і з'явився Інтернет, який відомий кожному.

В 1960 році, коли вперше виникла ідея P2P, з метою обміну файлами між дослідницькими установами США була створена мережа ARPANET [10]. Мережа складалася з кількох хостів, і кожен хост вважався однаково важливим. ARPANET виявилася успішною мережею, де кожен вузол мережі міг як запитувати, так і поставляти інформацію. Хоча ARPANET вважалася успішною мережею, що використовує простий механізм маршрутизації, він не був самоорганізованим і не мав можливості забезпечити будь-які засоби для маршрутизації на основі контексту чи вмісту.

Наступним кроком P2P мереж стала мережа Napster. Napster – пірингова мережа для обміну файлами [12]. Діаграма мережі Napster наведено на рис. 1.4.

Хоч мережа і є одноранговою, в ній присутній центральний сервер, його задача була збирати інформацію про файли, які зберігаються на комп'ютерах користувачів. Коли користувач хотів завантажити файл, він посилав запит на

центральний сервер, той в свою чергу повертав список клієнтів, в який є цей файл. Подальше отримання файлу проходило без участі центрального сервера, тобто клієнт, який хотів завантажити файл встановлював зв'язок напрям з клієнтом, в якого є цей файл і завантажував його. Головним недоліком такого підходу було те, що в мереж присутній центральний сервер, який після зростання кількості клієнтів став вузьким місцем через те, що на ньому знаходилось дуже багато інформації про файли користувачів і пошук потрібних файлів був дуже довгим [11]. Оскільки мережа Napster здебільшого використовувалась для завантаження музики, то постало питання авторських прав, оскільки клієнти, які скачують пісні не платили за це. Після того, як суд наказав вимкнути сервери мережі вона перестала існувати, оскільки вся інформація, необхідна для функціонування мережі припинила існувати.

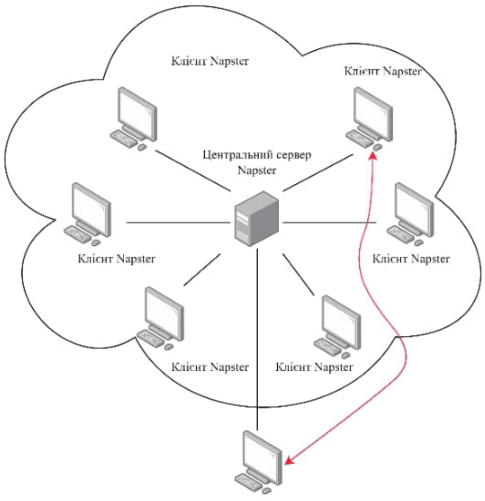


Рисунок 1.4 – Схема мережі Napster

Після припинення існування мережі Napster, було створено мережу Gnutella [12]. При створенні цієї мережі було враховано помилки попередника, тому ця мережа не містила центрального серверу. Діаграму мережі наведено на рис. 1.5.

Пошук потрібних файлі відбувався за наступним принципом:

– користувач під'єднувався до мережі Gnutella, а саме до деякої кількості клієнтів напряму;

– коли користувач хотів завантажити файл він надсилав запит на хости, до яких він приєднаний, якщо ж файл не знаходився, то ці хости надсилали запити до тих хостів, з якими вони з'єднані;

– якщо файл знаходився, то хости по-черзі надсилали дані для підключення до потрібного хосту;

– користувач, якому потрібно було завантажити файл встановлював підключення, скачував файл і переривав підключення.

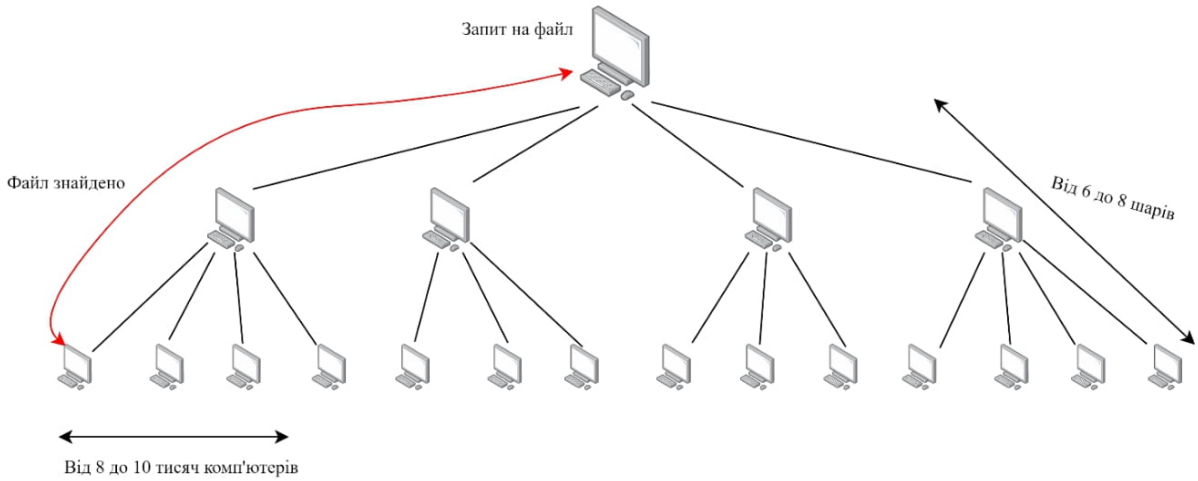


Рисунок 1.5 – Схема мережі Gnutella

Такий підхід дозволив існувати мережі без центрального серверу, але не обійшлось і без недоліків. У кожного запиту є час життя, тобто кількість пересилань до того, як запит стане неактивним. Це обмеження звужувало периметр пошуку, зазвичай один клієнт міг опитати приблизно десять тисяч комп'ютерів. Клієнт міг не знайти файл, хоч він і знаходився у мережі. Також час пошуку потрібної інформації міг займати багато часу. Оскільки комп'ютер користувача приймав участь у мережі Gnutella, то він також відповідав на запити і пересилав їх, що означало зменшення пропускної здатності мережі клієнта.

Підхід Gnutella став великою перевагою, адже він дозволив позбутися головного вузького місця мережі Napster, а саме центрального сервера. Якщо в цій мережі знаходиться хоча б два користувачі, то вона буде існувати і історія з відключенням центрального сервера не спрацює.

Хоч сама мережа і є законною, якщо ж користувачі починають обмінюватись файлами, які є під авторським правом, це порушення закону. Контролювати це неможливо, тому були придумані атаки на мережу Gnutella. Перевантаження мережі підробними пакети про пошук інформації, цей спосіб дозволяє практично вивести з ладу якусь частину мережі, оскільки клієнти в будь-якому разі будуть обробляти підроблені запити. Завантаження пошкоджених файлів, коли в мережу потрапляє такий файл, він починає передаватись між хостами і зупинити це неможливо.

Наступним етапом розвитку стала мережа KaZaa. Принцип її дії схожий на систему Gnutella, але замість того, щоб клієнти підключали один до одного, було використано протокол FastTrack [13]. Цей протокол ділить користувачів на вузли і супер-вузли. Супер-вузли обираються з користувачів, які мають найбільш стабільне та швидке з'єднання. Звичайні користувачі надсилають запити до супер-вузлів, а ті в свою чергу займаються пошуком потрібної інформації по мережі. Такий підхід дозволив позбутися обмеження у вісім тисяч опитаних користувачів, оскільки супер-вузли могли спілкуватись між собою, а це означає, що пошук був набагато точніше і надійніше. Але такий підхід не позбавив KaZaa усіх недоліків і атаки, які були актульними для Gnutella, залишились.

На сьогоднішній день найвідомішим протоколом P2P є BitTorrent. BitTorrent – це протокол для обміну файлами у P2P мережі [14]. BitTorrent розповсюджує файл, розділяючи його на шматки та розповсюджуючи їх між рівноправними користувачами. Зазвичай архітектура складається з наступних елементів [15]:

- трекер;
- статичний файл мета-інформації (торент файл);
- оригінальний завантажувач, seed;
- кінцевий завантажувач, leech;

Трекер BitTorrent – це серверне програмне забезпечення, яке централізовано координує передачу файлів між користувачами. Трекер не

містить копії файлу й лише допомагає одноранговим користувачам виявляти один одного.

Торент файл в основному містить закодовану інформацію щодо URL-адреси трекера, імені файлу та хешів фрагментів файлу для перевірки завантажених фрагментів файлу. Ці файли зазвичай створюються за допомогою клієнтського програмного забезпечення. Для створення торент-файлу потрібен список трекерів і вихідний файл.

Seed є звичайним користувачем мережі, який має у себе цілий файл. Він повинен продовжувати ділитися файлом, доки повна копія не буде розповсюджена серед завантажувачів. Поки є повна копія, спільно присутня серед користувачів, завантаження триватиме для всіх.

Піри без повної копії файлу відомі як leech. Вони отримають список користувачів із трекера, у яких є частини файлу, які потрібні. Потім leech завантажує потрібний фрагмент від одного з цих користувачів. Leech також може поширювати фрагменти, які він завершив завантажувати, ще до того, як він завершить завантаження всього файлу, це забезпечує сталу швидкість завантаження. Коли leech повністю завантажить файл, його можна назвати seed.

Протокол BitTorrent має недоліки, а саме [14]:

– простий спосіб поширення піратського/незаконного вмісту;

– неможливість змінити/оновити файл до новіших версій після розповсюдження торент-файлу;

– IP-адреси всіх пірів та інформація про файли, які вони завантажують, є загальнодоступними на трекерах;

– трекер є критично важливим компонентом, і якщо він виходить з ладу, це може порушити розповсюдження всіх файлів, які він відстежує.

Блокчейн – це спільний, незмінний реєстр, який полегшує процес запису транзакцій і відстеження активів у бізнес-мережі [16]. Актив може бути матеріальним (будинок, автомобіль, готівка, земля) або нематеріальним (інтелектуальна власність, патенти, авторські права, брендинг). Практично все,

що має цінність, можна відстежувати та торгувати в мережі блокчейн, що знижує ризики та скорочує витрати для всіх учасників. Ключові елементи блокчейну:

– технологія розподіленого реєстру, усі учасники мережі мають доступ до реєстру та його незмінного запису транзакцій. За допомогою цього реєстру транзакції створюються лише один раз, усуваючи дублювання, типове для традиційних бізнес-мереж;

– незмінні записи, жоден учасник не може змінювати або втручатися в транзакцію після її запису в реєстрі. Якщо запис транзакції містить помилку, необхідно додати нову транзакцію, щоб виправити помилку, і тоді обидві транзакції стануть видимими;

– розумні контракти, щоб зменшити час виконання транзакцій, набір правил, так званий смарт-контракт [17], зберігається в блокчейні та виконується автоматично. Смарт-контракт може визначати умови для переказу корпоративних облігацій, містити умови для оплати туристичної страховки та багато іншого.

Кожна транзакція, що відбувається, записується як «блок» даних. Такий блок показує рух активу. Блок даних може фіксувати інформацію будь-яку інформацію: хто виконав, що виконалось, коли виконалось, де виконалось, скільки виконалось і навіть різні умови – наприклад, температура харчового вантажу (рис. 1.6). Кожен блок пов'язаний з блоками до і після нього. Ці блоки утворюють ланцюжок даних, коли актив переміщується з місця на місце або право власності змінюється. Блоки підтверджують точний час і послідовність транзакцій, а також блоки надійно зв’язуються між собою, щоб запобігти зміні будь-якого блоку або вставці блоку між двома існуючими блоками.

Транзакції фіксуються разом у незворотній ланцюжок: блокчейн. Кожен додатковий блок посилює перевірку попереднього блоку і, отже, всього блокчейну. Це робить блокчейн неможливим для втручання, забезпечуючи ключову перевагу незмінності. Це усуває можливість втручання зловмисника і створює реєстр транзакцій, якому кожен учасник мережі може довіряти.

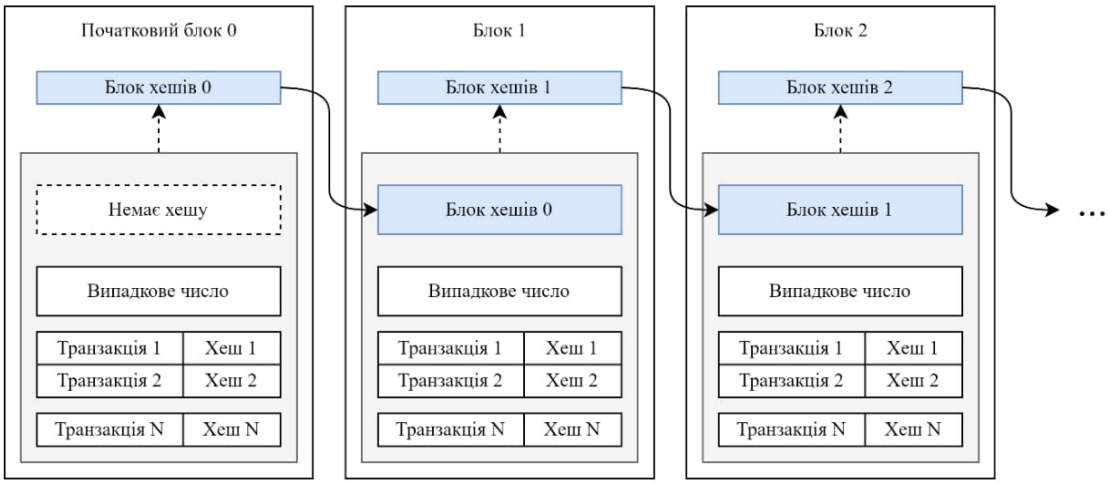


Рисунок 1.6 – Діаграма збережених блоків у мережі блокчейн

Децентралізовані фінанси (DeFi) – це нова фінансова технологія, яка базується на захищених розподілених реєстрах, подібних до тих, що використовуються в блокчейні.

Децентралізоване фінансування усуває посередників, дозволяючи людям, торговцям і підприємствам проводити фінансові операції за допомогою нових технологій. Через однорангові фінансові мережі DeFi використовує протоколи безпеки, підключення, програмне та апаратне забезпечення [18]. Усюди, де є підключення до Інтернету, люди можуть позичати, торгувати та позичати за допомогою програмного забезпечення, яке записує та перевіряє фінансові дії в розподілених фінансових базах даних. Розподілена база даних доступна з різних місць, оскільки вона збирає та агрегує дані від усіх користувачів і використовує механізм консенсусу для їх перевірки [19].

Децентралізоване фінансування усуває потребу в централізованій фінансовій моделі, дозволяючи будь-кому користуватися фінансовими послугами будь-де, незалежно від того, хто і де він знаходиться. Програми DeFi дають користувачам більше контролю над своїми грошима через особисті гаманці та торговельні сервіси, які обслуговують окремих осіб.

Пірингові мережі також можуть використовуватись електронної взаємодії державних електронних інформаційних ресурсів. У 2019 році у Україні було впроваджену систему «Трембіта». «Трембіта» – це децентралізована система

електронної взаємодії державних електронних інформаційних ресурсів [20]. Вона була створена на основі розробленої в Естонії системи «X-ROAD» [21].

Система «Трембіта» забезпечує:

– обмін електронними повідомленнями між інформаційними та/або інформаційно-телекомунікаційними системами суб'єктів владних повноважень;

– фіксацію часу надсилання та отримання електронних повідомлень;

– цілісність та автентичність електронних повідомлень та надає відомості, що дозволяють простежити історію руху електронних повідомлень;

– протоколювання дій суб'єктів владних повноважень під час обміну даними.

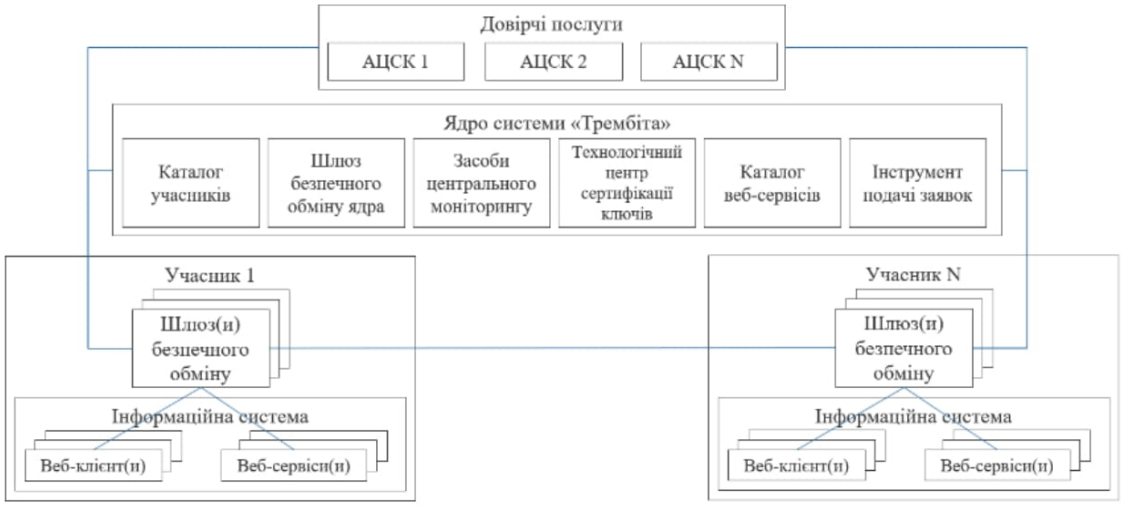


Рисунок 1.7 – Архітектура системи «Трембіта»

Така система реалізує архітектуру P2P, оскільки дані, які надходять до різних відомств зберігаються тільки в них, тобто для модифікації вони доступні тільки для цього відомства (рис. 1.7). Також, оскільки ця система забезпечує автентичність даних, інші відомства можуть бути впевнені, що дані, які вони запросили були створені саме тим, ким мали бути, а не підроблені.

Передача інформації відбувається лише за допомогою ядра «Трембіти», а це означає, що кожен клієнт може сам вирішувати яку архітектуру використовувати у своїй мережі.

1.3 Аналіз безпеки в P2P мережах

Однорангові мережі є чудовим джерелом інформації, розваг і спілкування. Більшість популярних мереж P2P засновані виключно на піратстві. Ось чому мільйони користувачів по всьому світу беруть участь у подібних мережах, навіть незважаючи на те, що в цих мережах є переважно незаконні та захищені авторським правом матеріали.

Таким чином, зловмисники можуть підробити файли, якими вони діляться, або зв'язок між вузлами. Оскільки P2P-мережі за своєю суттю залежать від однорангових вузлів, зловживання довірою між одноранговими вузлами може призвести до наслідків безпеки.

На табл. 1.2 наведено типи атак на P2P мережу та їх приклади.

Таблиця 1.2 – Типи атак на P2P мережу та їх приклади

Тип атаки	Приклад атаки
Атака на мережу	– забруднення – підробка – пропуск – людина посередині – перевірка вмісту – викрадення особи чи даних
Атака на протоколи маршрутизації	– затемнення – атака «Сивіли» – відтік – відображення ідентифікатора – отруєння мережі
Атака на пірів	– відмова в обслуговуванні / розподілена відмова в обслуговування (DoS/ DDoS)

Атака забруднення: атака забруднення відбувається, коли зловмисник навмисно вставляє небажані дані та надсилає непридатну інформацію іншому одноранговому вузлу. Одержувач може пересилати спам і забруднювати мережу в геометричній прогресії. Ця атака знижує якість обслуговування та потенційно втрачає пропускну здатність мережі. Методи, засновані на репутації [22], є найсильнішим методом боротьби з атаками забруднення, а також внесенням до чорного списку та використанням хеш-перевірки та шифрування.

Атака підробки: у цій атаці зловмисник підробляє дані та порушує їх цілісність і конфіденційність. Очевидно, що перевірка дайджесту повідомлення разом з асиметричними ключами для підпису повідомлення та симетричною криптографією для збереження конфіденційності можуть бути використані для запобігання цьому типу атак [23].

Атака відмови: Атака відмови відбувається, коли одноранговий пристрій відмовляється отримати повідомлення. Знову ж таки, криптографічні процедури як підписи повідомлень є найкращими методами для вирішення цієї вразливості [23].

Атака пропуску: коли одноранговий вузол не може переслати потік даних іншим одноранговим вузлам. У цій атаці виявити шкідливий вузол дуже важко. Ця бездіяльність може навіть скомпрометувати мережу P2P. Було продемонстровано, що автентифікація однорангових вузлів за допомогою підписаних підтверджень разом із моніторингом і занесенням у чорний список конкретного шкідливого однорангового вузла є відповіддю на виявлення конкретного зловмисника [23].

Атака "людина посередині": зловмисник під час атаки "людина посередині" перехоплює комунікації між двома вузлами; діє як обидві сторони та отримує доступ до даних, якими дві сторони планували обмінятися. Центральний довірений орган, який зазвичай не існує в більшості мереж P2P, потрібен для автентифікації кожного вузла. Крім того, впровадження механізму шифрування для захисту конфіденційності повідомлень, що обмінюються, є ще

однією дієвою практикою для виявлення та уникнення атаки «людина посередині» [24].

Перевірка вмісту: перевірка дійсності отриманого документа може не вважатися фактичною атакою. Однак у децентралізованій системі поки що немає гарантії, що однорангові розподіляють ресурс, який вони гарантують. Насправді нехтування перевіркою отриманого вмісту спричинило поширення шкідливого програмного забезпечення в мережах P2P [24]. Методи, засновані на репутації [25], є одними з найдієвіших методів боротьби з атаками забруднення, а також внесенням до чорного списку та використанням перевірки хешу та шифрування.

Викрадення особи чи даних: системи P2P зазвичай використовуються користувачами з обмеженою інформацією про безпеку ПК. Таким чином, їхні комп'ютери та записи піддаються підвищеному ризику атак із боку просунутих користувачів. Зловмисники можуть зловживати запитами недосвідчених користувачів, щоб допомогти їм отримати доступ до конфіденційної інформації. У більшості випадків атаки призводять до витоку всієї системи, паролів або будь-якої іншої конфіденційної інформації.

Атака «Сивіли»: ця атака є, мабуть, найскладнішою проблемою для вирішення в децентралізованих мережах P2P. Ця атака була вперше описана в 2002 році [26]. Під час атаки «Сивіли» єдиний зловмисний вузол створює багато фальшивих ідентифікацій вузла та діє як різний окремий фізичний вузол у системі. Ці підроблені особи називаються Сивілами. Іншими словами, зловмисник намагається отримати велику кількість підроблених однорангових ідентифікаторів у розподіленій хеш-таблиці (DHT) [27]. Отже, він міг би надсилати неправдиву інформацію та суттєво контролювати мережу. Наприклад, якщо зловмисник може вибрати свій ідентифікатор довільно, він може призначити собі набір ідентифікаторів ближче до певних ресурсів у P2P мережі. У більшості структурованих мереж P2P вузли вибирають випадкові ідентифікатори, коли вони входять у мережу, і система P2P не контролює призначення ідентифікаторів, що дозволяє зловмиснику отримати скільки завгодно ідентифікаторів і використовувати їх для компрометації мережі. Однак

P2P на основі лінійного діофантового рівняння (LDE) обмежує кількість логічних адрес на один тип ресурсу [28]. Ця техніка не дозволяє користувачам отримати різні підроблені ідентифікатори. Атаці «Сивіли» можна запобігти різними підходами. Перш за все, кількість ідентифікаторів користувачів на одного фізичного члена має бути обмежена. Одним із підходів є генерація хеш-ідентифікатору, використовуючи IP-адресу вузла. Ця техніка не є надійною, оскільки зловмисник може підробити багато IP-адрес, щоб отримати більше ідентифікаційних даних. Аутентифікація однорангового пристрою в мережі за допомогою логічного ідентифікатора є ефективним механізмом проти атаки «Сивіли», який інтегровано в P2P на основі LDE [29]. Використання відкритих і закритих ключів разом із надійним локальним центром розподілу ключів у P2P на основі LDE робить його несприйнятливим до будь-якого зловмисного вузла, який запитує отримання кількох підроблених ідентифікаторів.

Атака затемнення: атаки затемнення передбачають ізоляцію зловмисником певного користувача або вузла в одноранговій мережі (рис. 1.8). Під час виконання атаки затемнення зловмисник намагається перенаправити вхідні та вихідні з’єднання цільового користувача з його законних сусідніх вузлів на вузли, контрольовані зловмисником, тим самим ізолюючи ціль у середовищі, яке повністю відокремлене від фактичної мережевої активності.

Атака відтоку: ще одна важлива атака, яка, на диво, мало досліджена, це атака відтоку. Відтік є основною проблемою в кожній мережі P2P через імпульсивність однорангових користувачів. Піри можуть постійно приєднуватися до системи та залишати її. Часте приєднання та відхід однорангових користувачів, що визначається як відтік, створює вразливість у мережах P2P. Обслуговування таблиць маршрутизації завдяки незалежному приєднанню та виходу однорангових вузлів може зайняти дуже багато часу та ресурсів. Отже, оскільки рівень відтоку збільшується, система P2P загалом може зіткнутися з серйозними проблемами та не зможе відповісти на будь-який запит на приєднання чи пошук. Таким чином, зловмисник може скористатися ефектом

відтоку, спричинивши прибуття та відхід тисяч вузлів за короткі цикли та послабивши інфраструктуру P2P.

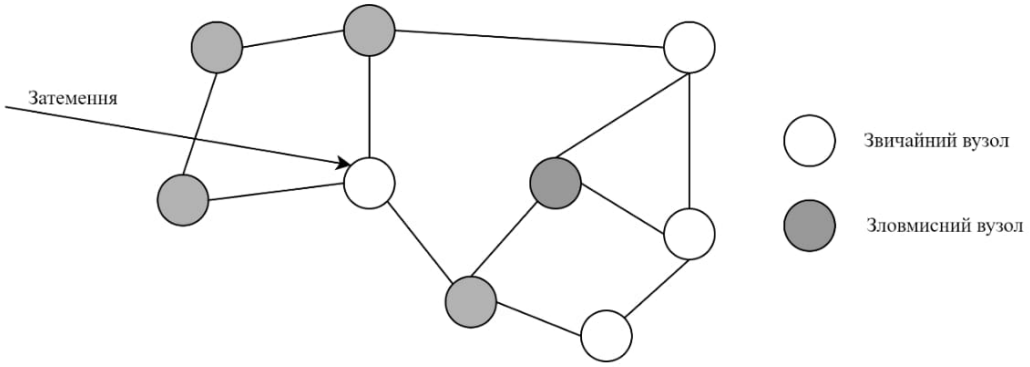


Рисунок 1.8 – Схема роботи атаки затемення

Отруєння мережі: це в основному підхід до введення непотрібних даних у структуру P2P, отже, отруєння. Через потребу P2P-систем актуалізувати адміністрування запитів у DHT або єдиному реєстрі, стає можливим вводити численні набори запити до файлу, таким чином перешкоджаючи системі. Ці помилкові набори запитів вплинуть на тривалість відповідей, а також можуть призвести до недійсних результатів. Атаки типу отруєння мережі можна поділити на 2 види:

– отруєння індексу: підроблені дані впроваджуються в списки, які вказують на мету IP-адреси та номер порту. Коли користувач намагається отримати доступ до даних, видаються підроблені дані пошкодженого файлу;

– отруєння таблиці маршрутизації: тут зловмисник використовує потребу всіх однорангових вузлів P2P, щоб підтримувати певну умову спрямування пов’язаних поточних однорангових вузлів. У структурі DHT, наприклад, кожна однорангова таблиця матиме певну кількість сусідів, задану тією ж кількістю системних вузлів. Підхід, який використовують зловмисники, полягає в створенні фальшивих сусідів у таблиці маршрутизації всіх користувачів. Це призводить до того, що ціль атаки має багато запитів на з’єднання одночасно. У багатьох випадках протоколи маршрутизації P2P матимуть інструмент, який використовується для виключення застарілих вузлів із таблиці маршрутизації.

Розподілена відмова в обслуговуванні (DDoS): це атака, яка може призвести до того, що вся мережа або активи, що містяться в мережі, стануть недоступними для клієнтів. Ці атаки можна пояснити як діяльність зі збору розсилок мережевих вузлів, щоб їх можна було використати проти однієї жертви. Зловмисники можуть використати ідею опитування систем P2P, щоб перевантажити систему та, як наслідок, пошкодити функціонування мережі [30]. DDoS-атаки набули значного значення в рамках P2P, оскільки будь-який вузол може працювати як маршрутизатор. Наприклад, зловмисний вузол у центрі системи може фактично діяти як маршрутизатор і перенаправляти запити, таким чином перевантажуючи інший вузол, визначений як цільовий. Зловмисні вузли можуть завдати шкоди через неправильне використання функцій системи. У багатьох випадках це призводить до повного колапсу мережі [31].

Атака переповнення запитів: ця атака може статися в неструктурованих P2P, таких як Gnutella [12]. У неструктурованих мережах P2P для пошуку користувач формує запит із ключовим словом пошуку та передає його своїм сусідам. Одержувачі співвідносять ключове слово зі своїми ресурсами, якщо вони знаходять збіг, повідомлення з відповіддю на запит надсилаються назад відправнику, що містить інформацію про те, як завантажити ресурс. Одноранговий вузол, який запитав файл, завантажує його безпосередньо. У цій атаці шкідливі вузли (які можуть бути багатьма одноранговими через атаку «Сивіли») генерують запити та переповнюють мережу якомога більше.

Атака TCP SYN Flooding Attack: Ця атака відбувається, коли TCP-рівень переповнюється 3-сторонніми запитами рукостискання. Ця атака є різновидом атаки на відмову в обслуговуванні, і вона може бути націлена на будь-який одноранговий пристрій системи або навіть у більшому масштабі може бути націлена на мережу P2P. Ціллю цієї атаки є вичерпування ресурсів системи шляхом надсилання великої кількості запитів на встановлення з’єднання.

1.4 Формалізація вимог та постановка задачі

Розроблювана інформаційна технологія орієнтована на використання в IT компаніях різної величини. Дані, що передаються можуть бути різними, починаючи з даних, які не несуть ніякої цінності і закінчуючи договорами, контрактами і комерційною таємницею. І чим цінніші дані, тим ретельніше потрібно їх захищати. P2P архітектура для цієї задачі набагато краще, аніж стандартна клієнт-серверна, оскільки такі дані не зберігаються на одному сервері, а на девайсах співробітників. Така мережа може чітко контролюватись, адже для доступу до таких даних потрібно підтвердження користувача, який вже знаходиться в мережі, тобто зловмиснику потрібно проробити важку роботу для доступу до цих даних.

Для досягнення поставленої мети необхідно:

- проаналізувати мережі передачі даних;
- проаналізувати існуючі пірингові мережі;
- розробити модель інформаційної технології;
- розробити алгоритми функціонування інформаційної технології;
- реалізувати програмний засіб на основі розроблених моделей та алгоритмів;
- зробити висновки про виконану роботу;

Програмний засіб має відповідати наступним вимогам, відповідно до вихідних даних розробки:

- працювати на мобільних операційних системах Android та IOS;
- передавати дані на основі технології peer-to-peer;

Для реалізації програмного застосунку було запропоновано використання мови програмування Dart, а для графічного інтерфейсу – фреймворку Flutter.

Отже, в розділі було проведено аналіз архітектур мереж передачі даних, розглянуто основні атаки на пірингові мережі, а також сформовані задачі, які необхідні для досягнення поставленої мети.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

2.1 Модель інформаційної технології

Інформаційна технологія захищеного обміну даними на основі піринговых мереж складається з таких процесів:

- процес автентифікації;
- процес створення ключів шифрування;
- процес створення унікального ідентифікатора користувача;
- процес зберігання даних в локальну базу даних (БД);
- процес встановлення з’єднання між користувачами;
- процес передачі даних.

На рисунку 2.1 наведено основні складові інформаційної технології захищеного обміну даними на основі піринговых мереж.

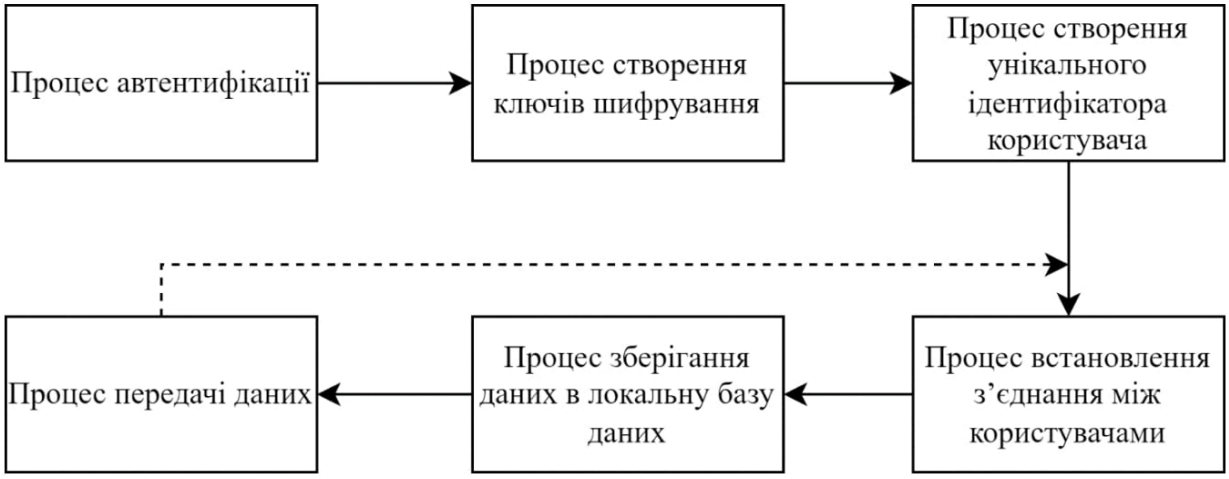


Рисунок 2.1 – Структура інформаційної технологіі захищеного обміну даними на основі піринговых мереж

Процеси спроектовані для виконання окремих задач, результат яких може використовувати наступним процесами для обробки і виконання їх задач.

Процес автентифікації потрібен для того, щоб ідентифікувати особу в мережі. Він полягає у введені користувачем логіну та паролю. Логін – це певний набір алфавітно-числовий набір, що ідентифікує користувача. Пароль – конфіденційний набір символів, букв або цифр, який при поєднанні з логіном

дає змогу однозначно ідентифікувати користувача системи. При стандартній клієнт-серверній архітектурі логін і пароль зберігаються в захищеній базі даних, а коли починається процес автентифікації – вони порівнюються, якщо вони співпадають, то користувачу дозволяється вхід в систему. У розроблюваній інформаційній технології роль логіну і паролю інша. Схема процесу автентифікації наведена на рис. 2.2.

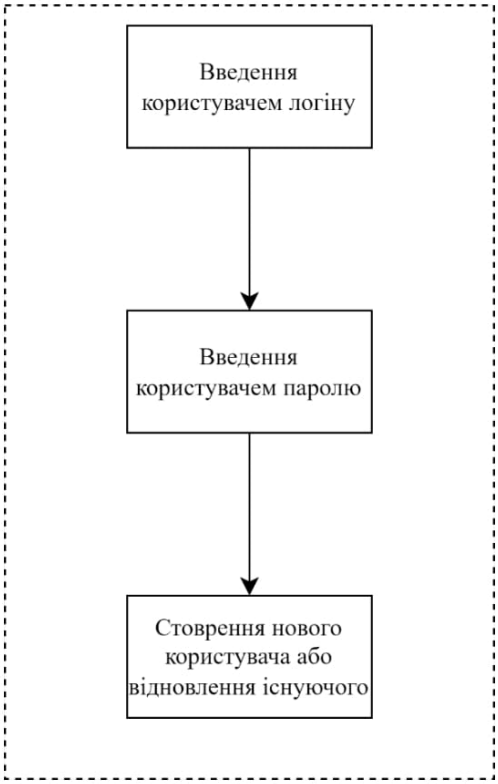


Рисунок 2.2 – Схема процесу автентифікації

Модель процесу автентифікації має наступний вигляд:

$$A = \{U, P\},$$

де U – логін користувача, P – пароль користувача, A – результат проходження процесу автентифікації.

Оскільки пірингова технологія передбачає збереження даних на одному пристрої, то у користувача є вибір – почати процес, як новий користувач або ж експортувати дані зі старого акаунту. Якщо ж користувач вводить логін і пароль, то він отримує доступ до системи, але на їх основі створюються ключі шифрування, які є підтвердження того, що це саме той користувач, за якого себе

видає. Якщо ж користувач обирає експорт даних, то він має ввести ті самі дані, які вводив коли реєструвався, оскільки саме на основі саме цих даних буде зроблено відновлення даних користувача.

Після проходження процесу автентифікації, починається процес створення ключів шифрування користувача. Ці ключі використовуються як для шифрування даних, що передаються по незахищеній мережі, так і для ідентифікації користувача. Спочатку створюється секретний ключ користувача, за допомогою якого користувач може розшифрувати повідомлення, зашифроване відкритим ключем. Відкритий ключ створюється на основі секретного, але за його допомогою можна тільки зашифрувати дані, прочитати їх не можна (рис. 2.3). Комбінований ключ обчислюється з секретного та відкритого ключів за допомогою спеціальної функції.

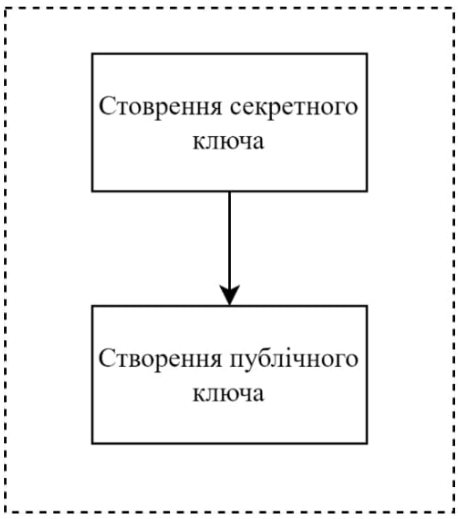


Рисунок 2.3 – Схема процесу створення ключів шифрування

Модель процесу створення ключів шифрування можна описати наступним чином:

$$SK = g_{SK}(A),$$
$$PK = g_{PK}(SK)$$

де SK – секретний ключ, g_{SK} – функція геренації секретного ключа, $A(U, P)$ – логін та пароль користувача, отримані на етапі автентифікації, PK – публічний ключ, g_{PK} – функція генерації публічного ключа.

Процес створення унікального ідентифікатора користувача потрібен для того, однозначного визначення користувача в системи. Він використовується під час встановлення з’єднання між користувачами, а також для додавання до списку контактів. Процес створення унікального ідентифікатора наведено на рисунку 2.4.

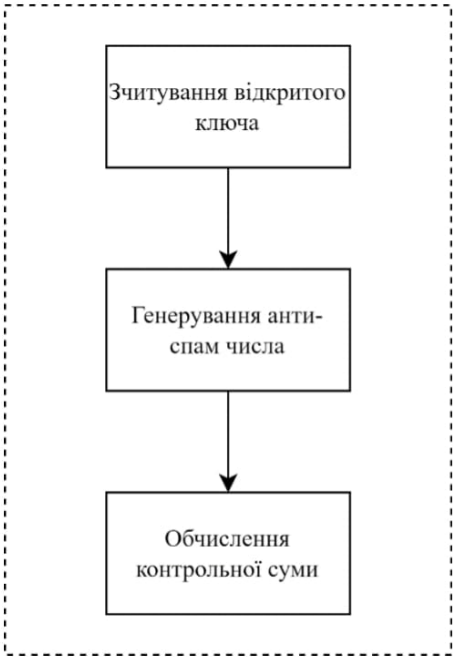


Рисунок 2.4 – Схема процесу створення унікального ідентифікатора

Цей ідентифікатор складається з трьох частин, а саме:

- довгостроковий публічний ключ;
- анти-спам число;
- контрольна сума.

Модель процесу створення унікального ідентифікатора можна описати наступною формулою:

$$UID = PK||N||S,$$

$$N = rand(4),$$

$$S = \sum_{i=1}^{n} b_i \oplus b_{i+1},$$

де UID – унікальний ідентифікатор користувача, PK – публічний ключ, N анти-спам число, S – контрольна сума, $||$ – операція конкатенації, $rand(4)$ – функція генерування чотирьох-байтового псевдовипадкового числа, b – байти ідентифікатора, n – кількість байт ключа.

Довгостроковий публічний ключ – це ключ, який створюється на основі секретного ключа користувача.

Анти-спам число – це число або набір чисел, встановлених одноранговим вузлом. Воно використовується, щоб запобігти розповсюдженню спаму у мережі дійсними запитами на додавання у контакти. Це гарантує, що лише користувачі, які бачили ідентифікатор однорангового партнера, можуть надіслати йому запит. Таке число може дозволити користувачам використовувати різні ідентифікатори і навіть змінювати ідентифікатори. Це може бути корисно, якщо зловмисник дізнався ідентифікатор користувача і надсилає йому велику кількість запитів на додавання до списку контактів. Зміна анти-спам числа зупинить вхідну хвилю спаму запитів на додавання до списку контактів без будь-яких негативних наслідків для списку контактів користувачів. Наприклад, якщо користувачу доведеться змінити свій відкритий ключ, щоб запобігти отриманню запитів, це позначатиме, що їм доведеться фактично відмовитися від усіх своїх поточних контактів, оскільки контакти прив’язані до відкритого ключа. Анти-спам число взагалі не використовується після того, як друзі додали один одного, що означає, що його зміна не матиме жодних негативних наслідків.

Контрольна сума обчислюється шляхом виняткової диз’юнкції (XOR) між кожною парою поряд-стоячих байтів відкритого ключа та анти-спам числа.

Процес встановлення з’єднання між користувачами починається з того, що один з користувачів надсилає запит на додавання до контактів іншому користувачу. Запит складається з трьох частин, а саме:

- ідентифікатора відправника;
- повідомлення;

Передавання публічного ключа є основною задачею процесу встановлення з’єднання, адже все подальше спілкування між двома користувачами відбувається з використанням цього ключа, тобто користувач до списку контактів додає саме ключ (рис. 2.5).

Математична модель процесу виглядає наступним чином:

$$C(A|I) = s(PD, UID),$$

де $C(A|I)$ – результат виконання запиту на встановлення з’єднання, A – підтвердження запиту, I – ігнорування запиту, PD – корисне навантаження, UID – унікальний ідентифікатор користувача, s – функція надсилання запиту.

Отримані запити на дружбу передаються клієнту, очікується, що клієнт побачить повідомлення із запиту на встановлення з’єднання і прийме рішення, чи хоче він його додавати чи ні.

Запити приймаються шляхом додавання ключа до списку контактів, який міститься у запиті, і відхиляються шляхом простого ігнорування.

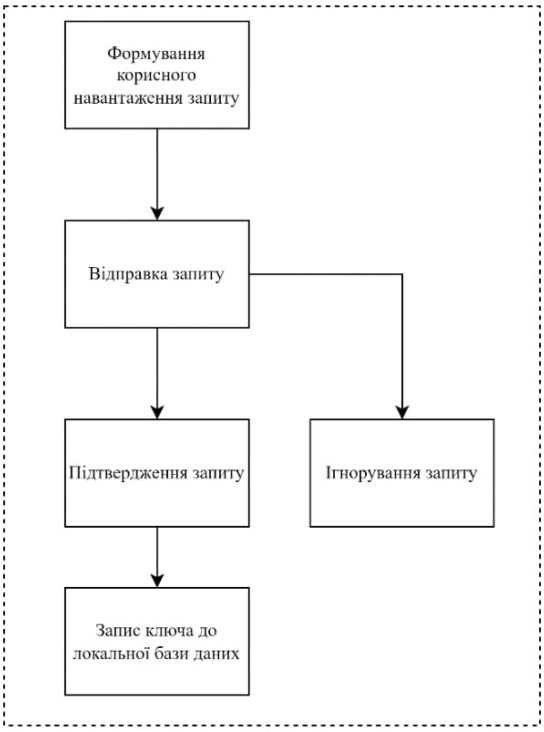


Рисунок 2.5 – Схема процесу встановлення з’єднання між користувачами

Оскільки інформаційна технологія розробляється на основі пірингових мереж, це означає, що дані не мають зберігатись будь-де, окрім пристрою, на

якому було пройдено процес автентифікації і у клієнта, який отримав ці дані. Тобто під час спілкування, дані передаються на пряму між клієнтами і зберігаються в локальній БД. Оскільки пристрій користувача також може бути схильним до зламу, то передбачено шифрування даних, що передаються або приймаються. Тобто процес зберігання даних в локальну БД зводиться до того, щоб перед надсиланням даних або після прийому, зашифрувати їх і зберегти для подальшого використання.

Процес передачі даних відбувається між клієнтами напряму, завдяки ключам, які були отримані під час процесу встановлення з'єднання. Для зручного знаходження користувачі, у них є псевдоніми, як обираються самими користувачами.

Рисунок 2.6 – Схема процесу передачі даних

Процес передачі даних можна описати наступними виразами:

$$DT = D_{1,\dots,n} \cup C(A) \cup PS \cup PE,$$

$$PS = CN \cup CID,$$

$$PE = (CID),$$

де DT – результат процесу передача даних, $C(A)$ – підтвердження встановленого з'єднання, $D_{1,\dots,n}$ – потік повідомлень для передачі, PS – пакет, що сигналізує про початок спілкування, PE – пакет, що сигналізує про завершення спілкування, CN – назва з'єднання, CID – ідентифікатор з'єднання.

Коли з'єднання між користувачами встановлено, то надсилається спеціальний пакет, котрий сигналізує про початок спілкування. Припиняється спілкування у двох випадках, коли зникає з'єднання між клієнтами або ж коли від співрозмовника отримано спеціальний пакет, котрий сигналізує про кінець спілкування. Дані надсилаються, використовуючи пряме з'єднання з співрозмовником (рис. 2.6).

Отже, в загальному інформаційну технологію захищеного обміну даними на основі піринговых мереж можна представити наступним чином:

$$IT = \{A, SK, PK, UID, C(A|I), DT\}$$

В результаті було розроблено модель інформаційної технології для захищеного обміну даними на основі піринговых мереж.

2.2 Розробка архітектури системи

Архітектура системи, що реалізує інформаційну технологію для захищеного обміну даними на основі піринговых складається з декількох модулів, які з'єднуються в одне ціле і користуються даними один від одного.

Елементи розроблюваної інформаційної технології (рис. 2.7):

- операційна система (ОС);
- модуль автентифікації;
- криптографічний модуль;
- модуль взаємодії з базою даних;
- модуль обміну даними;

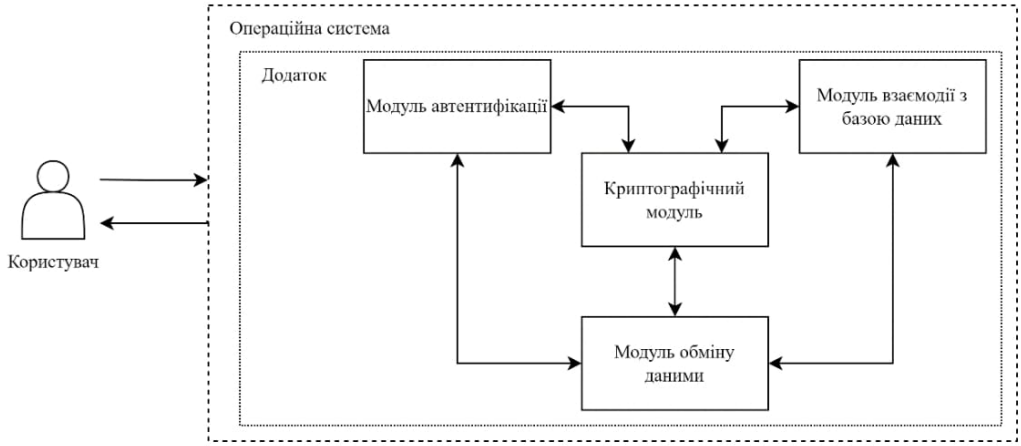


Рисунок 2.7 – Загальна архітектура системи

Для більшого розуміння необхідно детальніше розглянути кожен модуль.

Для функціонування додатку потрібно середовище, а саме система для роботи з файлами, система для роботи з мережею, тощо. Також додатку необхідно обробляти запити, які можуть бути створені поза додатком, саме операційна система дозволяє додатку.

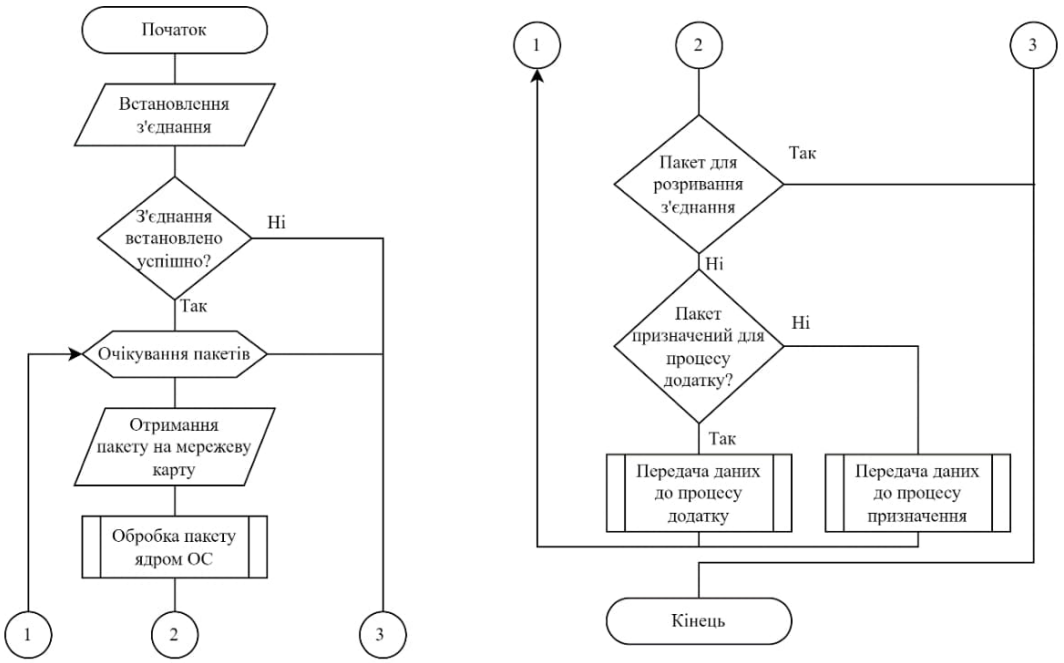


Рисунок 2.8 – Узагальнений алгоритм обробки мережевих запитів користувача операційною системою

Коли користувач заходить в додаток, перше, що він бачить – екран автентифікації. Після автентифікації можуть бути сформовані ключі

шифрування. На основі цих даних формуються секретні ключі, які в подальшому використовуються для відновлення акаунту, шифрування даних перед надсиланням та для шифрування даних, що зберігаються на пристрої користувача. Загальний алгоритм роботи додатку наведено на рис. 2.9.

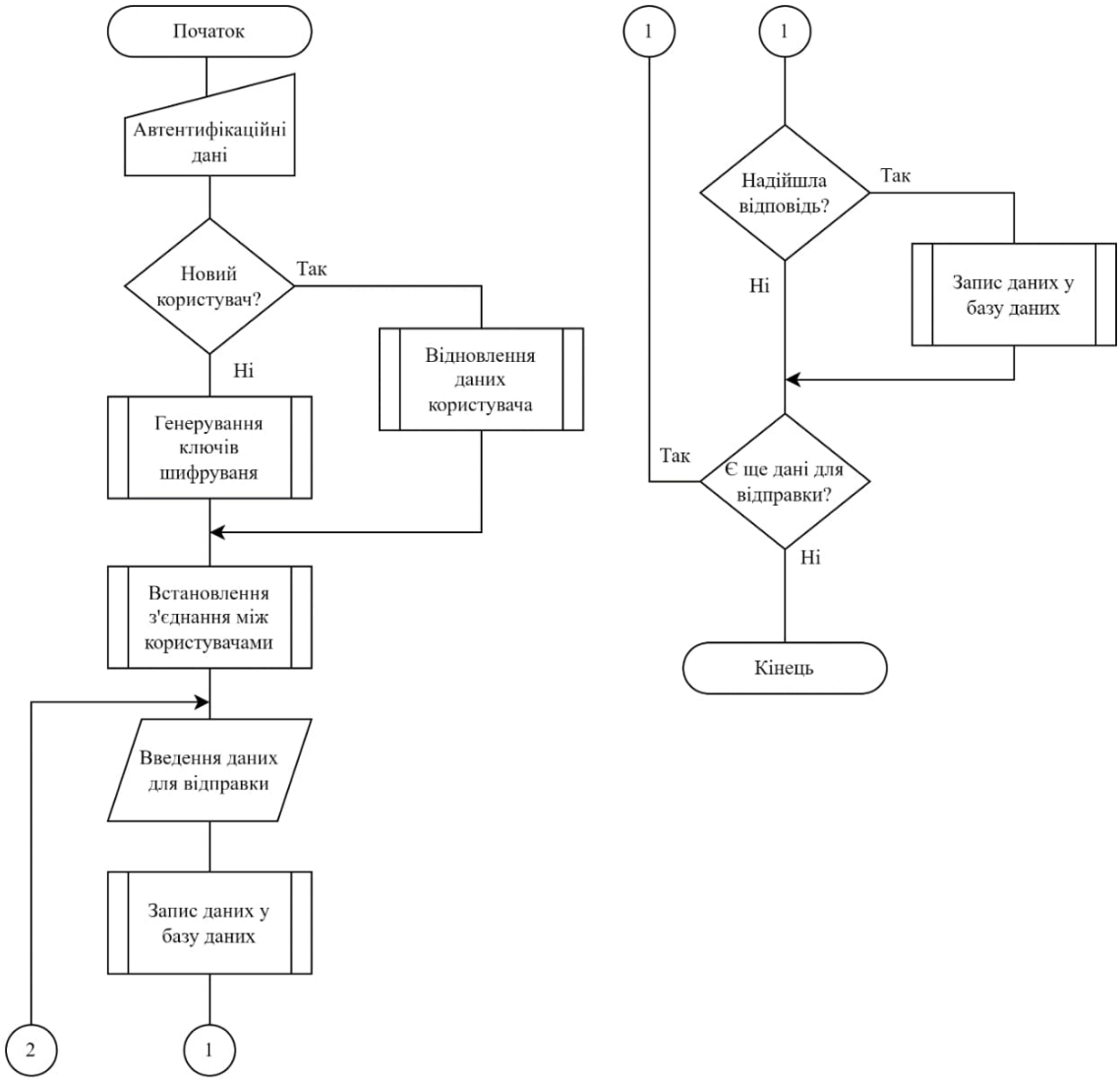


Рисунок 2.9 – Загальний алгоритм роботи додатку

Модуль автентифікації виконує декілька функцій, а саме:

– створення даних нового користувача;

– відновлення даних вже існуючого користувача.

Оскільки розроблювана інформаційна технологія зберігає дані тільки на стороні клієнта, в нього є можливість відновити акаунт на цьому пристрої, попередньо експортувавши його на минулому. Якщо ж користувач новий, то

йому пропонується ввести логін і пароль для автентифікації. Загальний алгоритм роботи модуля автентифікації наведено на рис. 2.10

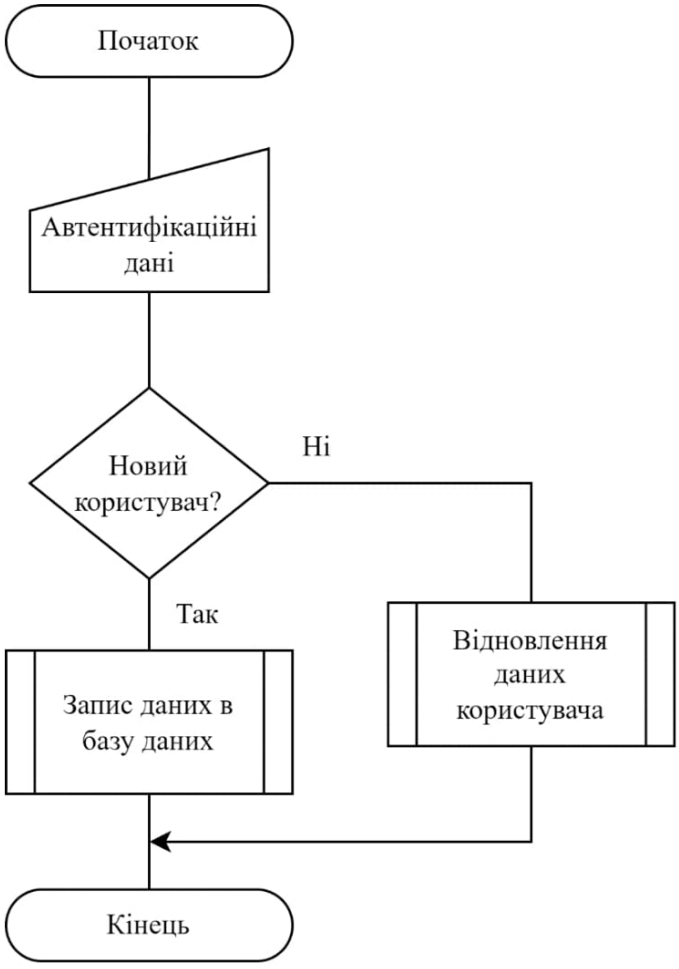


Рисунок 2.10 – Загальний алгоритм роботи модулю автентифікації

Після проходження процесу автентифікації, у разі якщо створився новий користувач, створюються ключі шифрування, які потрібні для передавання даних. Загальний алгоритм генерування ключів шифрування наведено на рис. 2.11. Також під час встановлення з'єднання користувачі обмінюються ідентифікаторами, які складаються з відкритого ключа, анти-спам числа (див. 2.1) та контрольної суми. Після встановлення з'єднання, користувачі повинні обрахувати спільний ключ, для того, щоб мати змогу розшифрувати дані повідомлення. Спільний ключ розраховується на основі секретного ключа користувача та відкритого ключа його співрозмовника [33]:

$$KP_1(SK_1, PK_2) = KP_2(SK_2, PK_1),$$

де KP_1, KP_2 – комбіновані ключі, SK_1, SK_2 – секретні ключі, PK_1, PK_2 – публічні ключі.

Рисунок 2.11 – Загальний алгоритм генерування ключів шифрування

Спільний ключ використовується для шифрування та розшифрування даних. Такий підхід дає змогу використовувати симетричний алгоритм шифрування з асиметричними ключами. Оскільки обчислення спільного ключа вимагає багато ресурсів, він генерується один раз при встановленні з'єднання та зберігається у базу даних. Загальну схему обміну ключами шифрування наведено на рис. 2.12.

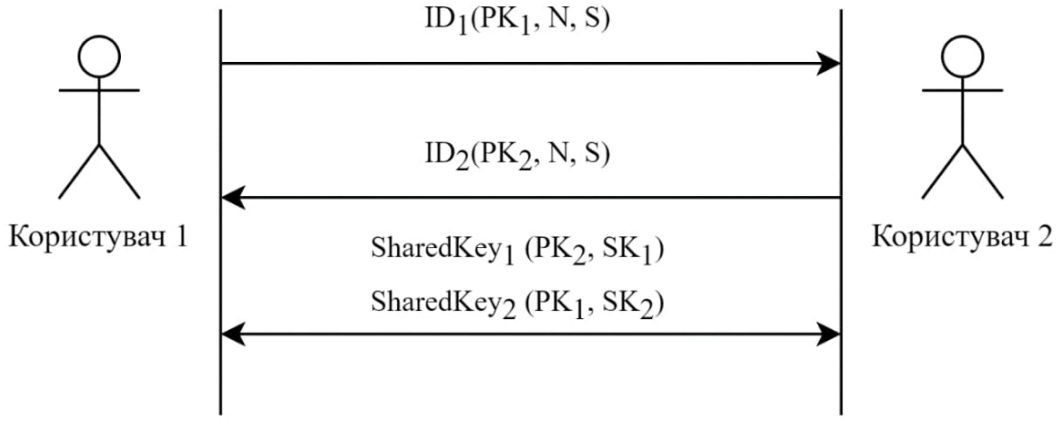


Рисунок 2.12 – Загальна схема обміну ключами шифрування

Розроблювана інформаційна технологія не передбачає автентифікації користувачів, реалізованої програмно, вирішення цієї проблеми відбувається наступним чином. Користувач може подивитись і звірити ідентифікатор користувача, який генерується за допомогою алгоритму UUID [34]. Якщо ідентифікатори співпадають, то такому контакту можна поставити спеціальну мітку, яка підтверджує особистість контакту.

Оскільки інформація зберігається на пристрої користувача, то її також необхідно захищати. Загальний алгоритм модулю взаємодії з базою даних наведено на рис

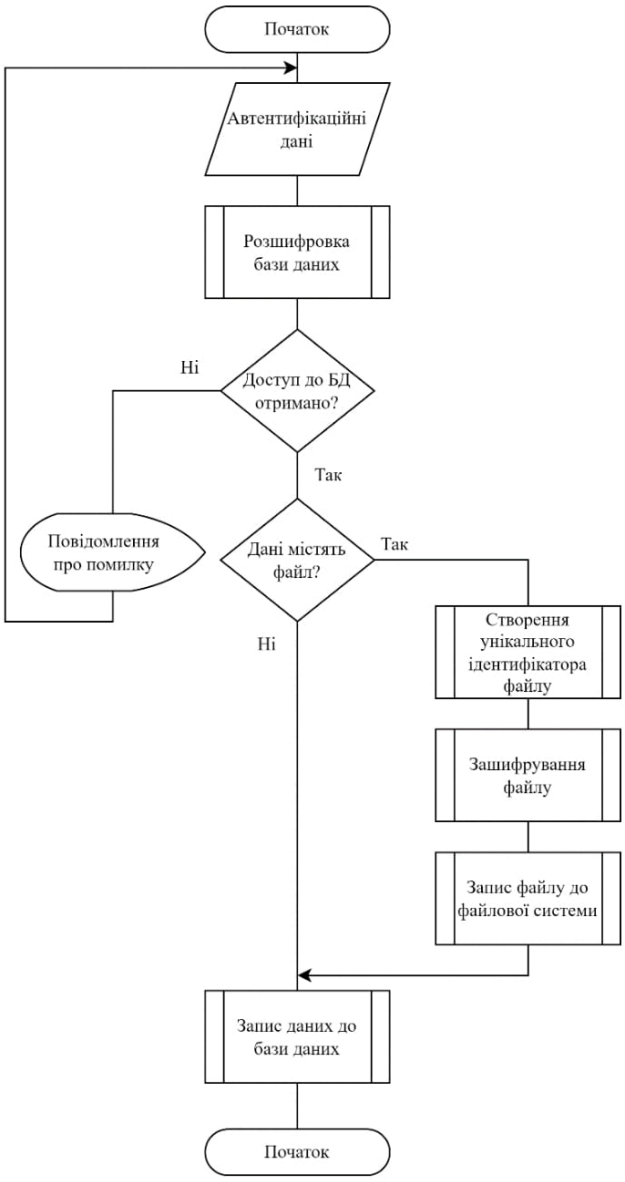


Рисунок 2.13 – Загальний алгоритм модулю взаємодії з базою даних

База даних захищена від доступу за допомогою паролю користувача, який він вводив під час автентфікації. Тобто замість того, щоб шифрувати кожне значення даних окремо, доступ до бази даних обмежено на вищому рівні, що позитивно впливає на продуктивність додатку, а також швидкість його роботи.

Будь-які дані, що передаються по мережі, будь-то файли, текст тощо, зберігаються а локальну базу даних, для того, щоб користувач міг доступ до них в будь-який момент. Але файли зберігаються трішки інакше, вони зберігаються на пристрої користувача в зашифрованому вигляді, при чому ідентифікатор такого файлу записується в базу даних.

Модуль обміну даними має зв’язок з усіма модулями системи. Коли користувач має намір передати дані іншому користувачу цей модуль зашифровує дані за допомогою спільного ключа шифрування, записує ці дані в базу даних та передає їх по мережі. Оскільки передача даних відбувається по бездротових канал зв’язку, радіус покриття яких може бути недостатнім щоб встановити з’єднання, то передбачено, що клієнти можуть виступати у ролі проміжних точок доступу. Схема передачі даних по мережі наведена на рис. 2.14.

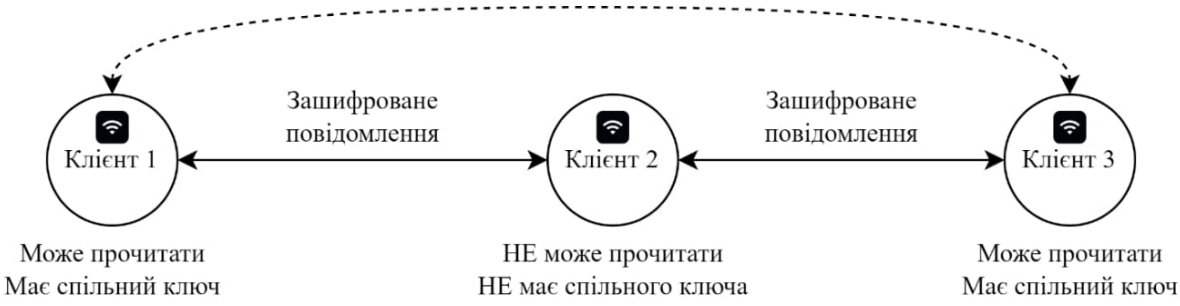


Рисунок 2.14 – Схема передачі даних по мережі з використанням проміжних клієнтів у ролі транспорту

Клієнти, які виступають у ролі проміжних точок доступу не можуть прочитати дані, що надсилаються між кінцевими точками, оскільки вони зашифровані за допомогою спільного ключа. Також, щоб надіслати дані клієнту, який виступає у ролі транспорту не потрібно проходити процес обміну ключами, якщо він не знаходиться у списку контактів.

2.3 Розробка модулю шифрування

Оскільки розробка орієнтується на передавання даних по мережі, при чому ці дані можуть бути такими, що потребують захисту, то у розроблюваній системі передбачено шифрування даних, що передаються. Для реалізації функцій шифрування необхідно обрати шифр, виходячи з того, що додаток має працювати на мобільних платформах. Оскільки технічні характеристики мобільних пристроїв зазвичай набагато слабкіші, аніж на персональних комп'ютерах, то одними з найважливіших параметрів є швидкість та ступінь захищеності даних.

Розширений стандарт шифрування (AES) — це симетричний блоковий шифр, обраний урядом США для захисту секретної інформації [35].

AES включає три блокові шифри:

– AES-128 використовує 128-бітний ключ для шифрування та дешифрування блоку повідомлень.

– AES-192 використовує 192-бітний ключ для шифрування та дешифрування блоку повідомлень.

– AES-256 використовує 256-бітний ключ для шифрування та дешифрування блоку повідомлень.

Кожен шифр шифрує та розшифровує дані блоками по 128 біт за допомогою криптографічних ключів 128, 192 та 256 біт відповідно. Симетричні шифри, використовують один ключ для шифрування та дешифрування. Відправник і одержувач повинні знати і використовувати той самий секретний ключ. Для шифрування інформації використовується 10 раундів для 128-бітних ключів, 12 раундів для 192-бітних ключів і 14 раундів для 256-бітних ключів. Раунд складається з кількох етапів обробки, які включають заміну, транспонування та змішування вхідного відкритого тексту для перетворення його на кінцевий вихідний зашифрований текст.

Перше перетворення в шифрі шифрування AES – це підстановка даних за допомогою таблиці підстановки. Друге перетворення зсуває рядки даних. Третій

змішує стовпчики. Останнє перетворення виконується для кожного стовпця з використанням іншої частини ключа шифрування. Довші ключі потребують більшої кількості раундів.

Переваги шифру AES:

– Безпека. Конкуруючі алгоритми повинні були оцінюватися за їхньою здатністю протистояти атакам порівняно з іншими представленими шифрами. Міцність безпеки мала вважатися найважливішим фактором змагань.

– Вартість. Алгоритми-кандидати, які планувалося випустити на глобальній, неексклюзивній та безоплатній основі, мали оцінювати на ефективність обчислення та пам'яті.

– Реалізація. Фактори, які слід враховувати, включали гнучкість алгоритму, придатність для реалізації апаратного чи програмного забезпечення.

Не дивлячись на всі переваги шифру AES, він вийшов повільним для виконання без апаратного прискорення. Для того, щоб прискорити шифрування методом AES в сучасні процесори додають спеціальний блок, який пришвидшує цю процедуру.

Дослідження атак на шифрування AES тривали з моменту завершення розробки стандарту в 2000 році. Дослідники знайшли кілька потенційних способів атакувати шифрування AES:

– У 2009 році дослідники виявили можливу атаку за пов'язаним ключем. Цей криптоаналіз намагався зламати шифр, вивчаючи, як він працює з використанням різних ключів. Атака пов'язаного ключа виявилася загрозою лише для неправильно налаштованих систем AES [36].

– У 2009 році відбулася атака з відомим ключем проти AES-128. Для визначення структури шифрування використовувався відомий ключ. Однак атака було успішною лише на восьми-раундову версію AES-128, а не на стандартну 10-раундову, що робить загрозу відносно незначною [36].

На рисунку 2.15 наведено загальну схему роботи шифрів сімейства AES.

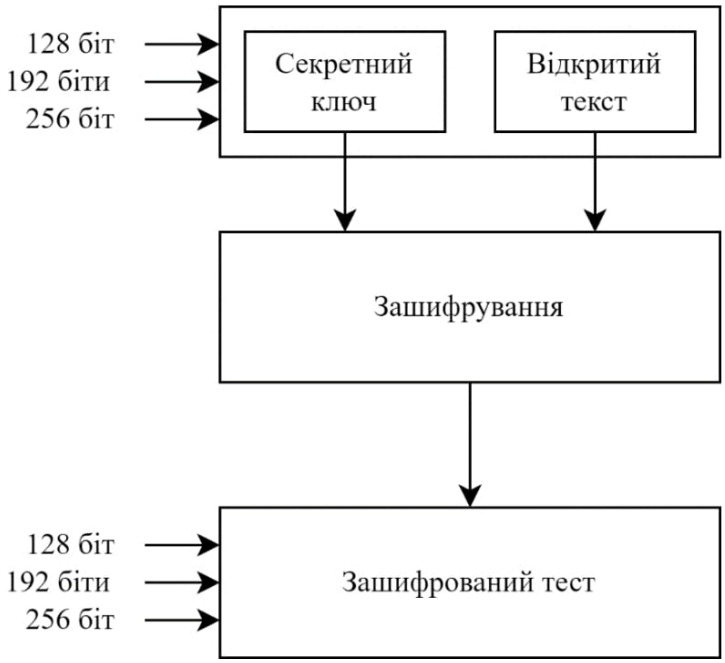


Рисунок 2.15 – Загальна схема роботи шифрів сімейства AES

Основним ризиком для шифрування AES є атаки по бічних каналах. Замість того, щоб намагатися атакувати грубою силою, атаки з бічних каналів спрямовані на те, щоб зібрати витік інформації із системи. Однак атаки побічних каналів можуть зменшити кількість можливих комбінацій, необхідних для атаки AES грубою силою. Атаки по бічних каналах включають збір інформації про те, що робить обчислювальний пристрій під час виконання криптографічних операцій, і використання цієї інформації для зворотного проектування криптографічної системи пристрою. Ці атаки можуть використовувати інформацію про час, наприклад, скільки часу потрібно комп'ютеру для виконання обчислень: електромагнітні витоки, звукові підказки, оптична інформація, наприклад, з камери з високою роздільною здатністю, щоб виявити додаткову інформацію про те, як система обробляє шифрування AES. В одному випадку була успішно використана атака побічного каналу для виведення ключів шифрування AES-128 шляхом ретельного моніторингу спільного використання шифром таблиць кешу процесорів. Загрозу від атак зі сторонніх каналів можна пом'якшити, запобігши можливим витокам даних. Крім того, використання методів рандомізації може допомогти усунути будь-який зв'язок між даними,

захищеними шифром, і будь-якими витоком даних, які можуть бути зібрані за допомогою атаки побічного каналу.

Оскільки довжина даних, що передаються наперед невідома, то кращим вибором буде використання потокового шифрування.

XChaCha20 — це 256-бітне потокове шифрування. Як і AES, він є симетричним і використовує єдиний ключ для шифрування та розшифровування даних. Оскільки шифр потоковий, то замість того, щоб ділити дані на блоки, XChaCha20 шифрує кожен біт даних окремо.

XChaCha використовує 192-бітний nonce. Nonce – довільне число, що може використовуватись тільки раз для того щоб зробити результат виконання операції (шифрування, хешування) унікальним. Наприклад, шифрування того самого рядка за допомогою того самого ключа призведе до того самого зашифрованого тексту. Щоб зробити шифрування унікальним, до повідомлення можна включити навмання вибраний одноразовий номер. Nonce також можна використовувати для захисту від атак повторного відтворення, хоча це робиться не у всіх випадках.

Шифр XChaCha використовується в IPsec, SSH, TLS 1.2, DTLS 1.2, TLS 1.3, QUIC, WireGuard [37]. Також його реалізовано в OpenSSH.

Менеджер паролів NordPass його використовує для шифрування збережених паролів з наступною аргументацією [38]:

– шифрування AES є швидким і безпечним, але воно демонструє деякі ранні ознаки потенційної можливості злому в майбутньому;

– він швидше, ніж AES-256. Він також приблизно в 3 рази швидший на платформах без апаратного забезпечення AES;

– він простіше, тобто технічних і людських помилок легше уникнути під час його впровадження;

– він не потребує апаратної підтримки;

– мобільні платформи повільно, але впевнено переходять на XChaCha20, тому найближчим він стане більш популярним.

Оскільки шифр XChaCha досить новий, то поки що не було проведено настільки велику роботу по криптоаналізу у порівнянні з AES. Але деякі спроби зламати шифр все ж були:

– Станом на 2022 рік немає успішних опублікованих атак на XChaCha, найкраща відома атака зламує 8 з 12 або 20 раундів [39].

– У 2005 році Paul Crowley повідомив про атаку на з орієнтовною складністю часу 2^{165}. Ця атака та всі наступні атаки базуються на усіченому диференціальному криптоаналізі [40].

– У 2006 році було повідомлено про атаку з орієнтовною складністю часу 2^{177}, а також про атаку за пов'язаним ключем з орієнтовною складністю часу 2^{217} [40].

RC4 – це потоковий шифр з різною можливою довжиною ключа. Хоча він є простим та швидким для реалізації, у RC4 було виявлено численні вразливості, що робить його небезпечним [41].

Ядро алгоритму RC4 складається з функції-генератора псевдовипадкових бітів, який видає потік бітів ключа (ключовий потік, гаму, послідовність псевдовипадкових бітів).

Алгоритм шифрування передбачає виконання таких дій:

– генерування послідовність бітів;

– виконання операції XOR згенерованої послідовності бітів з відкритим текстом.

Найуспішнішою атакою на RC4 є атака, яка була реалізована у 2015 році [42]. Вона була проведена на протокол TLS, який використовує RC4 для шифрування даних. Ця атака була виконана за 72 години, що є надзвичайно поганим показником захищеності даних, які зашифровані цим алгоритмом.

Алгоритм шифрування Rabbit був представлений у 2003 році на конкурсі eSTREAM [43]. Основним компонентом шифру є генератор бітового потоку, який шифрує 128 біт повідомлення за ітерацію. Функція змішування повністю базується на арифметичних операціях, доступних на сучасному процесорі, тобто

для реалізації шифру не потрібні додаткові апаратні засоби. Функція змішування заснована на арифметичному зведенні в квадрат, називається G-функцією. Також використовується операція ARX – логічне XOR з порозрядним обертанням фіксованих величин та додавання за модулем 2^{32}. G-функція, яка використовується в Rabbit – зведення 32-розрядного числа в квадрат для отримання 64-розрядного числа, а потім об'єднання лівої та правої половин цього квадратного числа за допомогою XOR для отримання 32-розрядного результату [44].

Атака з приблизною складністю 2^{247} була описана в 2006 році, але в 2008 складність вдалось зменшити до 2^{158}, це вдалось зробити за допомогою точного обчислення зміщення підблоків потоку ключів за допомогою швидкого перетворення Фур'є [45].

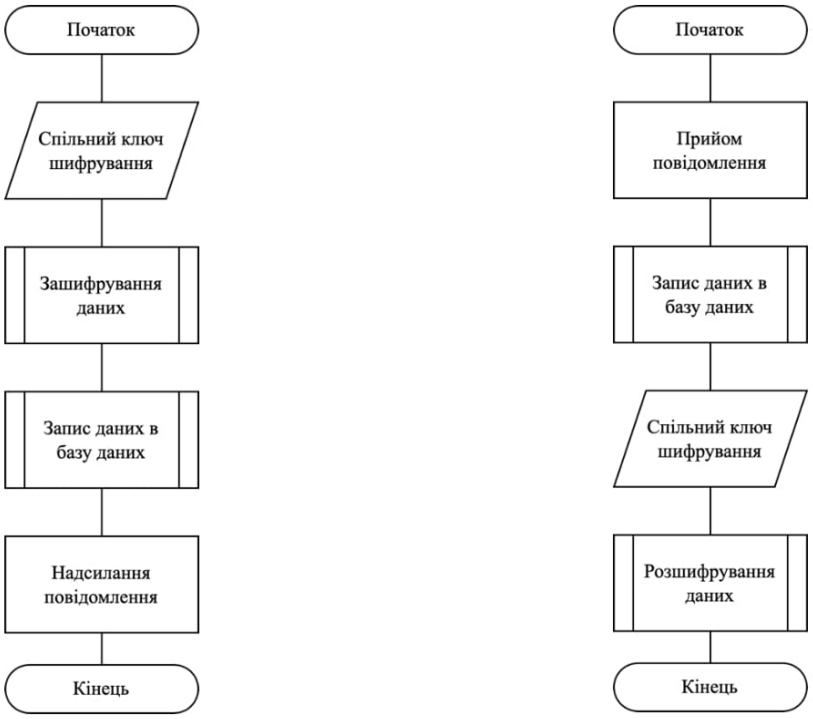
В таблиці 2.1 наведено порівняння алгоритмів шифрування RC4, Rabbit та XChaCha20.

Таблиця 2.1 – Порівняння алгоритмів шифрування

Алгоритм шифрування	Швидкість (циклів за байт)	Мінімальна обчислювальна складність
RC4	~ 7 – 13,9	~ 2^{13}
Rabbit	~ 3,7 – 9,7	~ 2^{158}
XChaCha20	~ 4,24 – 11,84	~ 2^{251} для восьми раундів

Беручи до уваги інформацію, що наведено вище, можна зробити висновок, що доцільно використати алгоритм шифрування XChaCha, оскільки його відношення швидкості до мінімальної обчислювальної складності є найбільшим. Цей шифр забезпечує високу стійкість даних до розкриття, простоту реалізації та швидкість виконання операції шифрування без додаткових апаратних потреб, що важливо для мобільних пристроїв.

Отже на основі проаналізованих даних, можна розробити модуль, який буде виконувати функцію зашифрування та розшифрування даних.



а) б)

Рисунок 2.16 – Загальний алгоритм процесу надсилання даних з використанням модулю шифрування (а – з боку передавача, б – з боку приймача)

На рис. 2.16 показано загальний алгоритм процесу надсилання даних з використанням модулю шифрування з двох сторін, з боку передавача та приймача. Передавач спочатку зчитує спільний ключ шифрування, який був обрахований під час встановлення з’єднання, за допомогою нього зашифровує дані, записує їх в базу даних та надсилає по бездротових каналах зв’язку. На стороні приймача алгоритм дій відрізняється тільки порядком виконання операцій.

У даному розділі було розроблено модель інформаційної технології, було наведено усі процеси, які в ній відбуваються, а також розроблені алгоритми її функціонування.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1 Обґрунтування вибору програмних засобів

Оскільки програмна реалізація інформаційної технології захищеного передавання даних на основі піринговых мереж передбачається у вигляді мобільного додатку, то необхідно обрати програмні засоби, які дозволяють виконати поставлену задачу.

Розробка мобільних додатків для платформ Android та iOS зазвичай виконується окрема і на різних мовах програмування. Такий підхід займає більше часу, а також ресурсів як людських, так і матеріальних. Для вирішення такої проблеми були створені спеціальні засоби, що дозволяють розробляти додатки для різних платформ з однією кодовою базою. Також це можуть бути не тільки мобільні додатки, а ще й веб-сайти, настільні програми тощо.

Для аналізу було використано статистику за 2019, 2020 та 2022 роки, яка формувалася на основі опитування більш ніж тридцяти тисяч розробників (рис. 3.1).

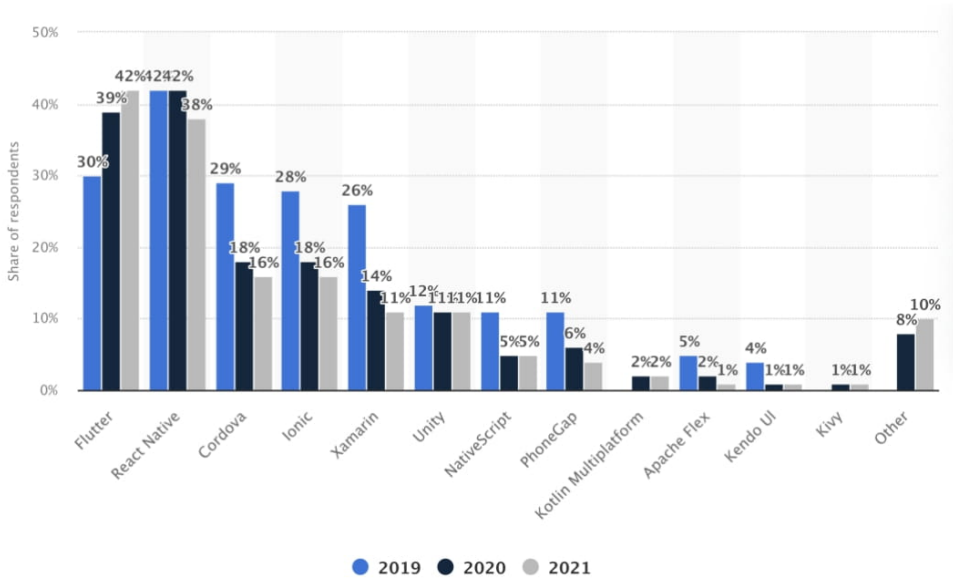


Рисунок 3.1 – Статистика популярності інструментів для кросплатформної розробки

З рисунку 3.1 видно, що три найбільш популярних засоби для кроплатформної розробки це Flutter, ReactNative та Cordova.

Apache Cordova — це платформа мобільної розробки з відкритим кодом [46]. Вона дозволяє використовувати стандартні веб-технології – HTML5, CSS3 і JavaScript для кросплатформної розробки. Програми виконуються в оболонках, націлених на кожну платформу, і покладаються на сумісні зі стандартами прив'язки API для доступу до можливостей кожного пристрою, таких як датчики, дані, стан мережі тощо. Cordova надає власну оболонку для компонентів додатка та використовує механізм візуалізації HTML – WebView, що є однією з основних причин зниження продуктивності розроблених додатків [47].

React Native — це фреймворк з відкритим кодом, запущений у 2015 році та написаний різними мовами, включаючи JavaScript, Swift, Objective-C, C++ і Python [48]. По суті, він надає всі інструменти та послуги для розробки високоякісних крос-платформних додатків на iOS, Android і Windows із можливістю пропонувати продуктивність рідного типу. ReactNative використовує спеціальні мости JavaScript для доступу до нативних функцій додатків, що погано впливає на продуктивність додатків, тому програми, що розроблені за допомогою ReactNative є значно повільнішими, аніж ті, що розроблені на рідних мовах платформ.

Flutter — це безкоштовний фреймворк для розробки інтерфейсу користувача з відкритим вихідним кодом, створений Google [49]. Цей фреймворк використовує мову програмування Dart, яка також розроблена компанією Google. Перевагою використання цієї мови є те, що вона безпосередньо компілюється в нативний код без використання мостів або сторонніх інструментів для цього. Для візуалізації використовується бібліотека двовимірної графіки Skia, що написана мовою C++. Skia дозволяє створювати дизайн різного ступеню складності, але при цьому він може бути відтвореним на різних розмірах екрану без втрати продуктивності.

Для розробки додатку було обрано фреймворк Flutter, оскільки він забезпечує високу продуктивність додатку, а також підтримку багатьох платформ, а саме iOS, Android, MacOS, Windows, Linux, Web. Це означає, що в майбутньому можна створити додаток, який реалізовуватиме однаковий функціонал на цих платформах з використанням однієї кодової бази.

В якості бази даних було обрано SembastDB. SembastDB – це полегшена NoSQL база даних, яка зберігається на пристрої користувача у спеціальному файлі. Цю базу даних реалізовано на мові Dart, що значно полегшує її використання з фреймворком Flutter. Всі дані записуються у спеціальний файл, який стискається для досягнення мінімального об’єму займаної пам’яті. Також ця БД надає інструменти для реалізації шифрування вмісту, тобто розробник може сам обрати та реалізувати алгоритми шифрування цієї бази даних.

3.2 Розробка програмного засобу

Для реалізації програмного застосунку використовується архітектура, яка наведена на рис. 3.2.

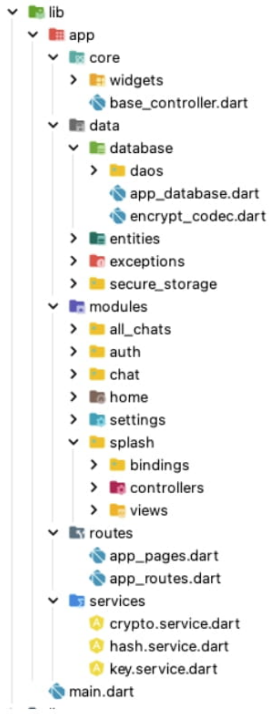


Рисунок 3.2 – Вигляд структури проекту додатку

В директорії lib міститься увесь код проєкту, файл main.dart є точкою старту програми. В директорії modules знаходяться всі екрани, які є у додатку. Один екран складається з трьох обов'язкових директорій, а саме:

– controllers – це директорія, яка містить контролер, в якому відбуваються всі дії з даними;

– view – це директорія, яка містить графічний інтерфейс, який залежить від даних, що формуються в контролері;

– bindings – це директорія, яка містить файл, що відповідає за залежності. Цей файл надає можливість контролеру або відображенню користуватись функціями, які можуть бути реалізовані в іншому класі;

Такий підхід дає змогу розмежувати дані та відображення, що позитивно впливає на швидкість розробки, а також розуміння коду.

В директорії data містяться файли, які відповідають за роботу з базою даних.

В директорії routes містяться файли, які відповідають за навігацію між екранами додатку.

В директорії services містяться спеціальні класи, які можуть використовуватись як залежності, наприклад методи генерації ключів шифрування, методи шифрування тощо. Ці методи можуть використовуватись будь-де у додатку.

Оскільки додаток має відображати всі дані в реальному часі, то необхідно використовувати спеціальні інструменти, які дозволять перемальовувати екран тоді, коли це потрібно для того, щоб відобразити нові дані. Для цієї задачі у фреймворкі Flutter є вбудовані інструменти, але їх використання не завжди є зручним і накладає деякі обмеження. Тому було створено багато бібліотек, які дозволяють виконувати цю задачу набагато простіше і надають більше контролю над процесом перемальовки даних. Однією з таких бібліотек є GetX. GetX – надзвичайно легке та потужне рішення для Flutter, він поєднує в собі високоефективне управління станом, інтелектуальне впровадження залежностей

і швидке й практичне керування маршрутами [50]. Робота з даними відбувається за допомогою обгорток типів, які надає ця бібліотека. Коли дані змінюються, то подається сигнал для інтерфейсу перемалюватись з новими даними. Після виконання коду, який наведено на рис. 3.3, до списку користувачів, з якими поточний користувач може встановити з'єднання додається новий і ці зміни відобразяться в графічному інтерфейсі.

```
final users = <User>[].obs;

void addUser(User user) {
  users.add(user);
}
```

Рисунок 3.3 – Код роботи з даними за допомогою бібліотеки GetX

Плагін flutter_nearby_connections надає доступ до роботи з Bluetooth та WiFi Direct, а також надає інструменти для встановлення пірингового з'єднання між користувачами [51]. Для встановлення з'єднання необхідно запустити сервіс, який відповідає за знаходження пристроїв користувачів у мережі. Коли новий пристрій знаходиться або зникає виконується функція, в яку параметром приходить список всіх пристроїв, з якими можна встановити з'єднання (рис. 3.4).

```
await nearbyService.init(
  serviceType: 'mpconn',
  deviceName: user.fullName,
  uuid: user.uuid,
  strategy: Strategy.P2P_CLUSTER,
  callback: (isRunning) async {},
);

devicesSubscription =
    nearbyService.stateChangedSubscription(callback: (devicesList) {
  devices.clear();
  devices.addAll(devicesList);
});
```

Рисунок 3.4 – Фрагмент коду запуску пірингового сервісу

Параметрами для того, щоб запустити сервіс є:

– ім’я пристрою – це ім’я яке вводить користувач, воно відображається у графічному інтерфейсі;

– унікальний ідентифікатор – це випадковий ідентифікатор, який потрібен для того, щоб точно визначити пристрій в мережі, оскільки ім’я може повторюватись;

– стратегія відповідає за те, яка топологія буде використовуватись для встановлення з’єднання. На рис. 3.4 видно, що використовується стратегіє P2P_CLUSTER, вона означає, що кожен з учасників мережі може як сам ініціювати з’єднання, так і прийняти його.

Для шифрування, хешування та генерації ключів шифрування використовується стандартна бібліотека crypto.

При реєстрації користувач вводить ім’я користувача, логін та пароль. На основі логіну та паролю генеруються ключі шифрування, які в подальшому використовуються для шифрування повідомлень, а також бази даних. У користувача є можливість відновити акаунт, якщо він вже користувався додатком, але при цьому йому потрібно обов’язково заздалегідь експортувати базу даних, яка зберігається на пристрої у вигляді спеціального файлу. Експортування відбувається за допомогою функцій, які надає база даних. При експортуванні бази даних, користувач може обрати локацію, куди потрібно зберегти БД, при цьому вона зашифрована, тобто прочитати її неможливо без знання логіну та паролю користувача.

Коли користувач хоче встановити з’єднання він обирає іншого користувача зі списку, після цього починається процедура обміну ключами шифрування та генерація спільного ключа за шифрування повідомлень. Фрагмент коду обробки повідомлень наведено на рис. 3.5.

Після обміну системними повідомленнями, які відповідають за обмін ключами, користувач може починати надсилати дані.

```
void _messageReceived(dynamic receivedData) {
  final data = receivedData as MessageEntity;
  switch(data.type) {
    case MessageType.startConnection:
      _sendPublicKeyToPeer(data.deviceId);
      break;
    case MessageType.receivePublicKey:
      _calculateSharedKey(data.startMessageData, data.deviceId);
      break;
    case MessageType.dataMessage:
      _handleDataMessage(data.dataMessage,data.deviceId);
      break;
    default:
      throw Exception('Unsupported message type');
  }
```

Рисунок 3.5 – Фрагмент коду обробки повідомлень

Перед надсиланням даних повідомлення зашифровується за допомогою спільного секретного ключа, який обраховується під час встановлення з'єднання та записується у базу даних. Фрагмент коду, який відповідає за надсилання повідомлення наведено на рис. 3.6

```
void sendData(String deviceId, String message, SecretKey sharedKey) async {
  final encryptedData = await CryptoService.base64Encrypt(sharedKey, message);
  await _saveMessageToDatabase(deviceId, encryptedData);
  nearbyService.sendMessage(deviceId, encryptedData);
}
```

Рисунок 3.6 – Фрагмент коду відправки даних користувачем

Коли користувач приймає дані процес схожий, але за одним виключенням, він повинен розшифрувати дані.

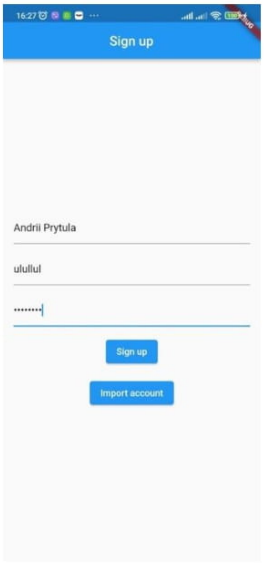
Оскільки кожне повідомлення записується до локальної бази даних, то коли користувач вийде з додатку, ці дані не втрачаються, але якщо ж користувач перевстановить додаток або ж вийде з акаунту, то вони будуть втрачені.

Вимогами для роботи додатку версії операційної системи Android 6 та вище, IOS 11 та вище. Також необхідний доступ до геолокації для точного визначення розташування пристрою користувача. Дані про геолокацію користувача використовуються тільки для роботи додатку і не записуються нікуди.

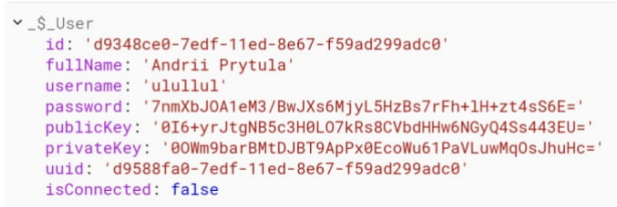
3.3 Тестування роботи програмного засобу

Тестування буде відбуватись на двох пристроях під керування операційної системи Android та за допомогою інструменту Logcat. Logcat — це інструмент командного рядка, який виводить системні повідомлення, коли пристрій видає помилку або надсилає повідомлення [52].

Для створення нового користувача необхідно ввести дані: ім’я користувача, логін та пароль. Після того, як користувач підтверджує введені дані, в локальну БД записуються дані користувача, при чому, генеруються ключі шифрування, пароль хешується, а також створюється унікальний ідентифікатор користувача. Результат тестування наведено на рис. 3.7.



а)

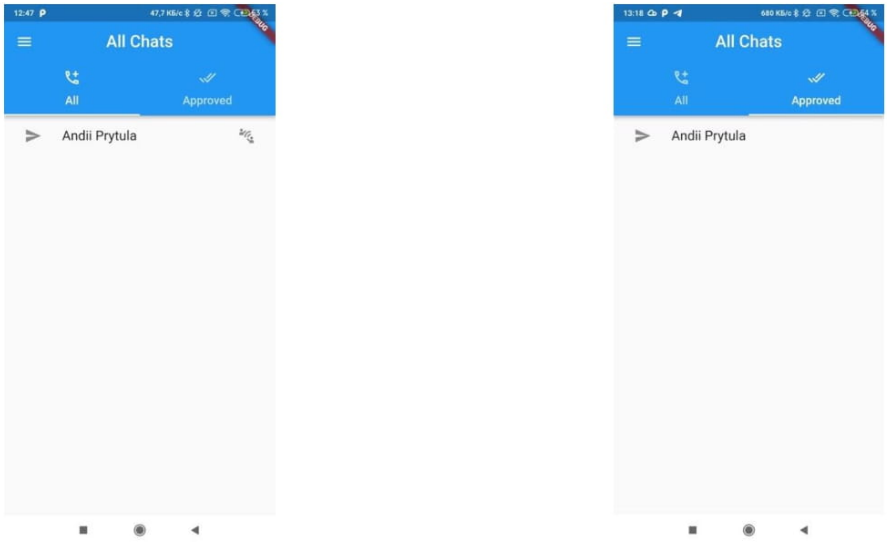


б)

Рисунок 3.7 – Результат тестування створення користувача (а – дані, введені користувачем, б – дані, які було записано до БД)

Коли користувач вже створений, він може обрати з ким встановити з’єднання зі списку користувачів (рис. 3.8). Якщо користувач, що приєднався до мережі не знайдений у локальній базі даних, то це означає, що з’єднання з ним ще не встановлювалось. Щоб мати змогу обмінюватись повідомленнями з цим користувачем необхідно почати процес обміну ключами шифрування, для цього

використовується кнопка праворуч від імені користувача. Якщо з'єднання було встановлено, то кнопка зникає і можна починати процес обміну даними.



а) б)

Рисунок 3.8 – Вигляд екрану з усіма користувачами мережі (а – до встановлення з'єднання, б – після встановлення з'єднання та підтвердження особи користувача)

Після того, як користувач обрав з ким встановити з'єднання починається процес обміну ключами шифрування і обрахунку спільного ключа. Після завершення цих операцій, ключ записується в БД. З рис. 3.9 видно, що спочатку користувач, який хоче ініціювати з'єднання надсилає свій публічний ключ, коли користувач, що приймає з'єднання отримує його, він в свою чергу надсилає свій. Після обміну ключами обраховується спільний ключ, і, як видно на рис. 3.9, обраховані ключі в результаті однакові.

```
12-18 21:15:20.959  1634  1920 I flutter : DL| SEND DATA: bPhMGNhJuCHA5CjxUfnIuQQqvNrfDUTDgIjc2NJGmjg=
12-18 21:15:23.501  1634  1920 I flutter : DL| DATA RECEIVED: 0I6+yrJtgNB5c3H0LO7kRs8CVbdHHw6NGyQ4Ss443EU=
12-18 21:15:24.129  1634  1920 I flutter : DL| SHARED KEY: MvP7BmHa9RgWFhLpSnivuBCvrpQOxr60Vx/AVTReuBY=
```

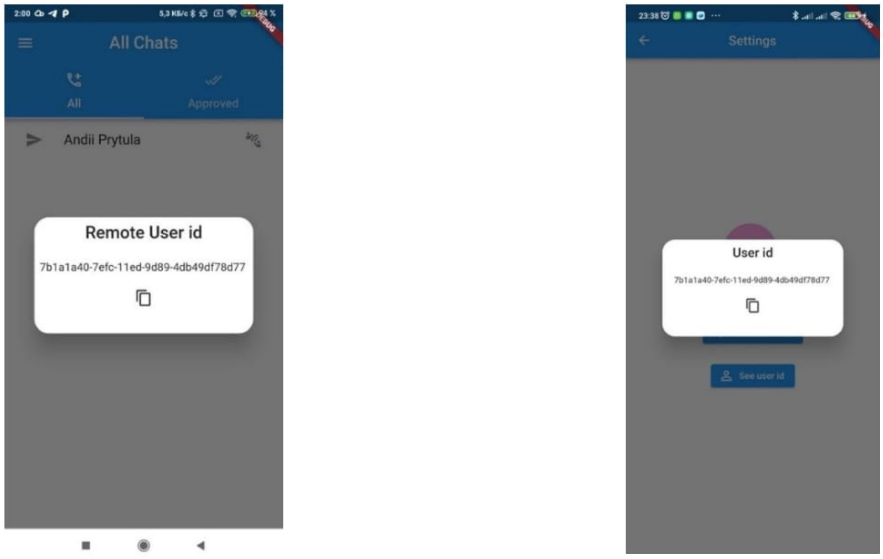
а)

```
12-18 21:15:18.664  9580  9616 I flutter : DL| DATA RECEIVED: bPhMGNhJuCHA5CjxUfnIuQQqvNrfDUTDgIjc2NJGmjg=
12-18 21:15:18.958  9580  9616 I flutter : DL| SHARED KEY: MvP7BmHa9RgWFhLpSnivuBCvrpQOxr60Vx/AVTReuBY=
12-18 21:15:20.978  9580  9616 I flutter : DL| SEND DATA: 0I6+yrJtgNB5c3H0LO7kRs8CVbdHHw6NGyQ4Ss443EU=
```

б)

Рисунок 3.9 – Результат обміну ключами шифрування (а – з боку ініціатора з'єдання, б – з боку приймача з'єдання)

Для перевірки користувача є спеціальний ідентифікатор, який генерується під час створення користувача і не може бути зміненим. Для того, щоб перевірити цей ідентифікатор, користувач, якого перевіряють повинен зайти на сторінку налаштувань та натиснути кнопку. Як видно на рис. 3.10, ідентифікатори, що показуються коли користувач знаходиться на екрані налаштувань та коли другий користувач обирає того ж користувача на екрані всіх чатів співпадають.



а) б)

Рисунок 3.10 – Вигляд ідентифікатора користувача (а – на віддаленому пристрої, б – на пристрої користувача)

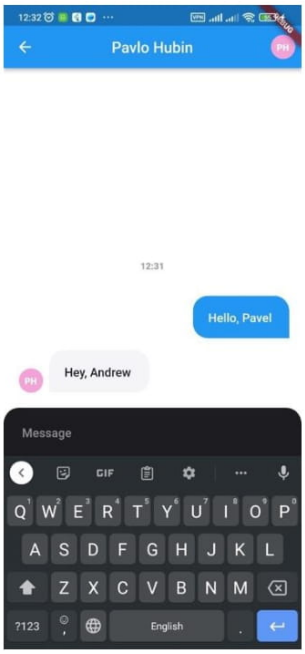
Дані передаються в зашифрованому вигляді, при чому вони шифруються спільним ключем, який був обрахований під час обміну ключами шифрування. На рис. 3.11 показано процес передачі зашифрованих даних від відправника, та розшифрування даних з боку приймача.

```
12-19 01:19:13.219 26276 26318 I flutter : DL| SEND ENCRYPTED DATADUfXmk981SzcHaF16bPmVboWrfJj
Y0S2WYrtWYYpzje75eC8gVWahczhJ4WPHF1zPAQG/w==
```

а)

```
12-19 01:32:27.364 17375 17499 I flutter : DL| ECRYPTED DATA RECEIVED: 2TGVPa7E06llHb5pMc/nL9AOjU4qVbL9+y
z9kw2fxzD7Jrm+zy8zbiodAv9uX5pYfjHs+w==
12-19 01:32:27.366 17375 17499 I flutter : DL| KEY FOR DECRYPT DATA: MvP7BmHa9RgWFhLpSnivuBCvrpQOxr60Vx/A
VTReuBY=
12-19 01:32:27.412 17375 17499 I flutter : DL| DECRYPTED DATA: Hello, Pavel
```

б)



в)

Рисунок 3.11 – Вигляд процесу передавання даних (а – зі сторони відправника, б – зі сторони приймача, в – в графічному інтерфейсі)

Якщо під час передачі даних повідомлення було змінене, то його неможливо розшифрувати, а також якщо розшифрування буде відбуватись за допомогою іншого ключа (рис. 3.12).

```
12-19 01:41:47.029 17375 17499 I flutter : DL| KEY FOR DECRYPT DATA: MvP7BmHa9RgWFhLpSnivuBCvrpQOxr60Vx/AVTReuBU=
12-19 01:41:47.063 17375 17499 I flutter : DL| ERROR: wrong parameters for decryption
```

Рисунок 3.12 – Спроба розшифрування даних з неправильним ключем

Як видно з рис. 3.12, при спробі розшифрувати дані з неправильним ключем виникає помилка.

У даному розділі було обрано програмні засоби, за допомогою яких реалізовувалась інформаційна технологія. Було реалізовано алгоритми функціонування, які були розроблені у попередньому розділі. Також було проведено тестування, за результатами якого можна стверджувати, що розроблений програмний засіб працює правильно.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота «Інформаційна технологія захищеного обміну даними на основі піринговых мереж» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

Метою проведення комерційного і технологічного аудиту дослідження за темою «Інформаційна технологія захищеного обміну даними на основі піринговых мереж» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної

діяльності. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 4.1 [53].

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 4.1

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуютьс я у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці (табл. 4.2).

За результатами розрахунків, наведених в таблиці 4.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 4.3 [53].

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	2	2	3
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	3	3	2
5. Ринкові переваги (експлуатаційні витрати)	2	2	2
6. Ринкові перспективи (розмір ринку)	2	2	2
7. Ринкові перспективи (конкуренція)	3	2	3
8. Практична здійсненність (наявність фахівців)	5	5	5
9. Практична здійсненність (наявність фінансів)	3	4	3
10. Практична здійсненність (необхідність нових матеріалів)	4	5	5
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	3	4
Сума балів	39	39	40
Середньоарифметична сума балів $СБ_с$	39,3		

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інформаційна технологія захищеного обміну даними на основі піринговых мереж» становить 39,3 бала, що, відповідно до таблиці 4.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

4.2 Розрахунок узагальненого коефіцієнта якості розробки

Окрім комерційного аудиту розробки доцільно також розглянути технічний рівень якості розробки, розглянувши її основні технічні показники. Ці показники по-різному впливають на загальну якість проектної розробки.

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення розрахуємо за формулою [54]:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i \quad , \tag{4.1}$$

де k – кількість найбільш важливих технічних показників, які впливають на якість нового технічного рішення;

α_i – коефіцієнт, який враховує питому вагу *і-го* технічного показника в загальній якості розробки. Коефіцієнт α_i визначається експертним шляхом і при цьому має виконуватись умова $\sum_{i=1}^{k} \alpha_i = 1$;

β_i – відносне значення *і-го* технічного показника якості нової розробки.

Відносні значення β_i для різних випадків розраховуємо за такими формулами:

– для показників, зростання яких вказує на підвищення в лінійній залежності якості нової розробки:

$$\beta_i = \frac{I_{нi}}{I_{ai}} , \tag{4.2}$$

де $I_{нi}$ та $I_{на}$ – чисельні значення конкретного *і*-го технічного показника якості відповідно для нової розробки та аналога;

– для показників, зростання яких вказує на погіршення в лінійній залежності якості нової розробки:

$$\beta_i = \frac{I_{ai}}{I_{нi}} ; \tag{4.3}$$

Використовуючи наведені залежності можемо проаналізувати та порівняти техніко-економічні характеристики аналогу та розробки на основі отриманих

наявних та проектних показників, а результати порівняння зведемо до таблиці 4.4.

Таблиця 4.4 – Порівняння основних параметрів розробки та аналога.

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований пристрій	Відношення параметрів нової розробки до аналога	Питома вага показника
Порівняльна швидкість передачі даних	бал	8	8,5	1,06	0,1
Надійність роботи мережі	%	90	95	1,06	0,25
Рівень захищеності даних	%	89	97	1,09	0,3
Дружність інтерфейсу	бал	4	9	2,25	0,15
Співвідношення ресурс/продуктивність	бал	5	8	1,6	0,2

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення складе:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i = 1{,}06 \cdot 0{,}1 + 1{,}06 \cdot 0{,}25 + 1{,}09 \cdot 0{,}3 + 2{,}25 \cdot 0{,}15 + 1{,}6 \cdot 0{,}2 = 1{,}36.$$

Отже за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,36 рази.

4.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Інформаційна технологія захищеного обміну даними на основі пірингових мереж», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

3.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп,

науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [53]:

$$З_о = \sum_{i=1}^{k} \frac{М_{ni} \cdot t_i}{Т_р}, \quad (4.2)$$

де k – кількість посад дослідників залучених до процесу досліджень;

$М_{ni}$ – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

$Т_р$ – середнє число робочих днів в місяці, $Т_р$=21 дні.

$$З_о = 16950{,}00 \cdot 42 / 21 = 33900{,}00 \; грн.$$

Проведені розрахунки зведемо до таблиці.

Таблиця 4.5 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	16950,00	807,14	42	33900,00
Інженер-розробник програмного забезпечення	16750,00	797,62	38	30309,52
Консультант (фахівець напряму "Кібербезпека")	17000,00	809,52	5	4047,62
Лаборант	6800,00	323,81	15	4857,14
Всього				73114,29

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Інформаційна технологія захищеного обміну даними на основі пірингових мереж» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \qquad (4.3)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{М_М \cdot К_і \cdot К_с}{Т_р \cdot t_{зм}} , \qquad (4.4)$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо *Мм*=6700,00 грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [54];

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ – середнє число робочих днів в місяці, приблизно $Т_р$ = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$$С_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}65 / (21 \cdot 8) = 72{,}38 \ грн.$$
$$З_{р1} = 72{,}38 \cdot 8{,}00 = 579{,}07 \ грн.$$

Таблиця 4.6 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка обладнання для розробки програмного забезпечення	8,00	2	1,10	72,38	579,07

Продовження таблиці 4.6

Підготовка робочого місця розробника програмного забезпечення	6,00	3	1,35	88,83	533,01
Інсталяція програмного забезпечення для захисту обміну даними	5,00	5	1,70	111,87	559,33
Компіляція програмних блоків	11,00	4	1,50	98,71	1085,76
Налагодження програмних блоків	4,00	5	1,70	111,87	447,46
Формування бази даних мережі	16,00	2	1,10	72,38	1158,14
Попереднє тренування системи	4,50	3	1,35	88,83	399,76
Всього					4762,53

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.5)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (73114,29 + 4762,53) · 11 / 100% = 8566,45 грн.

3.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.6)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (73114,29 + 4762,53 + 8566,45) · 22 / 100% = 19017,52 грн.

3.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Інформаційна технологія захищеного обміну даними на основі піринговых мереж».

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \quad (4.7)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, (K_j = *1,1 ... 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$$M_1 = 3,0 \cdot 225,00 \cdot 1,11 - 0 \cdot 0 = 749,25 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 4.7 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Папір офісний Mondi A4 80 г/м Smart Line OFFICE білий	225,00	3,0	0	0	749,25
Папір офісний кольоровий Spectra Color A4 80 г/м 10 кольорів по 10 аркушів	80,00	3,0	0	0	266,40

Продовження таблиці 4.7

Набір настільний 4-410 15 предметів	279,00	4,0	0	0	1238,76
Набір настільний 7015 - 08 SCHOLZ	351,00	2,0	0	0	779,22
Тонер IPM HP LJ P1005/1006/150 5/CANON LBP-3010/3020 (TSH87B) black	520,00	2,0	0	0	1154,40
Диск оптичний CD-RW	21,00	3,0	0	0	69,93
USB флеш накопичувач Transcend 64Gb JetFlash 700 (TS64GJF700)	290,00	1,0	0	0	321,90
Всього					4579,86

3.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Інформаційна технологія захищеного обміну даними на основі піринговых мереж», розраховуємо, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \qquad (4.8)$$

де $Н_j$ – кількість комплектуючих *j*-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих *j*-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ *= 1,1 ... 1,15*).

$К_в = 1 \cdot 2340,00 \cdot 1,11 = 2597,40$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.8 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Оперативна пам'ять: - оперативна пам'ять 12 ГБ - тип оперативної пам'яті DDR4	1	2340,00	2597,40
Графічний адаптер: - тип відеокарти вбудована - кількість ядер: 8	1	4360,00	4839,60
Маршрутизатор TP-Link ARCHER-C80	1	1799,00	1996,89
Всього			9433,89

3.3.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (4.9)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, (K_i = *1,10…1,12);*

k – кількість найменувань устаткування.

$$B_{спец} = 8999{,}00 \cdot 1 \cdot 1{,}1 = 9898{,}90 \text{ грн.}$$

Отримані результати зведемо до таблиці:

Таблиця 4.9 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Мобільний телефон Xiaomi Redmi Note 10S 6/128GB Onyx Gray	1	8999,00	9898,90
Мобільний телефон Apple iPhone Xs 64GB Gold (MT9G2)	1	23499,00	25848,90
Всього			35747,80

3.3.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.10)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$B_{прг} = 99,00 \cdot 1 \cdot 1,11 = 109,89$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 4.10 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Середовище розробки: - Android Studio	1	99,00	109,89
- XCode	1	105,00	116,55

Продовження таблиці 4.10

- Inteliij IDEA Community	1	95,00	105,45
Всього			331,89

3.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.11)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (56899,00 \cdot 2) / (2 \cdot 12) = 4741,58$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.11 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук Apple macbook air M1 2020 Процесор: - Apple M1 - кількість ядер: 8	56899,00	2	2	4741,58
Робоче місце інженера-розробника ПЗ	8700,00	5	2	290,00

Продовження таблиці 4.11

Пристрої передачі даних комутатор мережевий TP-Link	2780,00	2	2	231,67
Пристрій виводу інформації	6740,00	5	2	224,67
Оргтехніка	7250,00	4	2	302,08
Приміщення лабораторії	725000,00	25	2	4833,33
ОС Windows 10	8370,00	2	2	697,50
Прикладний пакет Microsoft Office 2016	7825,00	2	2	652,08
Всього				11972,92

3.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (4.12)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 6,20$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$$B_e = 0,04 \cdot 310,0 \cdot 6,20 \cdot 0,95 / 0,97 = 76,88 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 4.12 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук Apple macbook air M1 2020 Процесор: - Apple M1 - кількість ядер: 8 ядра	0,04	310,0	76,88
Робоче місце інженера-розробника ПЗ	0,15	310,0	288,30
Пристрої передачі даних комутатор мережевий TP-Link	0,01	320,0	19,84
Пристрій виводу інформації	0,50	25,0	77,50
Оргтехніка	0,62	5,0	19,22
Інше	0,10	300,0	186,00
Всього			667,74

3.3.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Інформаційна технологія захищеного обміну даними на основі піринговых мереж» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (4.14)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св} = 20\%$.

$В_{св}$ = (73114,29 + 4762,53) · 20 / 100% = 15575,36 грн.

3.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (4.15)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 35%.

$$B_{сп} = (73114,29 + 4762,53) \cdot 35 / 100\% = 27256,89 \text{ грн.}$$

3.3.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.16)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$$I_в = (73114,29 + 4762,53) \cdot 50 / 100\% = 38938,41 \text{ грн.}$$

3.3.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}\ , \qquad (4.17)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 115%.

$В_{нзв}$ = (73114,29 + 4762,53) · 115 / 100% = 89558,34 грн.

Витрати на проведення науково-дослідної роботи на тему «Інформаційна технологія захищеного обміну даними на основі піринговых мереж» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}\ . \qquad (4.18)$$

$В_{заг}$= 73114,29 +4762,53 +8566,45 +19017,52 +4579,86 +9433,89 +35747,80 + 331,89 + 11972,92 +667,74 +15575,36 +27256,89 +38938,41 +89558,34 = 339523,89 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}\ , \qquad (4.19)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 339523,89 / 0,9 = 377248,76 грн.

4.4 Розрахунок економічної ефективності науково-технічної розробки

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Інформаційна технологія захищеного обміну даними на основі піринговых мереж» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

Таблиця 4.13 – Збільшення кількості споживачів

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	1000	2000	2500	2000

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 15000 осіб;

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 8150,00 грн;

$\pm\Delta Ц_о$ – зміна вартості послуги від впровадження результатів, приймемо 397,32 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [53]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (4.20)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2022 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо $\rho = 40\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2022 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (397{,}32 \cdot 15000{,}00 + 8547{,}33 \cdot 1000) \cdot 0{,}83 \cdot 0{,}4 \cdot (1\text{-}0{,}18/100\%) = 3949440{,}13$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (397{,}32 \cdot 15000{,}00 + 8547{,}33 \cdot 3000) \cdot 0{,}83 \cdot 0{,}4 \cdot (1\text{-}0{,}18/100\%) = 8603287{,}64$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (397{,}32 \cdot 15000{,}00 + 8547{,}33 \cdot 5500) \cdot 0{,}83 \cdot 0{,}4 \cdot (1\text{-}0{,}18/100\%) = 14420597{,}04$ грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4 = (397{,}32 \cdot 15000{,}00 + 8547{,}33 \cdot 7500) \cdot 0{,}83 \cdot 0{,}4 \cdot (1\text{-}0{,}18/100\%) = 19074444{,}56$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (4.21)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0{,}27$;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП = 3949440{,}13/(1+0{,}27)^1 + 8603287{,}64/(1+0{,}27)^2 + 14420597{,}04/(1+0{,}27)^3 +$

$+ 19074444{,}56/(1+0{,}27)^4 = 3109795{,}38 + 5334049{,}01 + 7039990{,}59 + 7332245{,}82 = 22816$

080,79 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \qquad (4.22)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2,1;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 377248,76 грн.

$PV = k_{інв} \cdot ЗВ = 2{,}1 \cdot 377248{,}76 = 792222{,}40$ грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{4.23}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 22816080,79 грн;

PV – теперішня вартість початкових інвестицій, 792222,40 грн.

$$E_{абс} = ПП - PV = 22816080{,}79 - 792222{,}40 = 22023858{,}38 \text{ грн.}$$

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{4.24}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 22023858,38 грн;

PV – теперішня вартість початкових інвестицій, 792222,40 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+22023858{,}38/792222{,}40)^{1/4} = 1{,}32.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \tag{4.25}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,27.

$\tau_{мін}$ = 0,1+0,27 = 0,37 < 1,32 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інформаційна технологія захищеного обміну даними на основі піринговых мереж» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{4.26}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,32 = 0,76 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже, згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інформаційна технологія захищеного обміну даними на основі піринговых мереж» становить 39,3 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Також термін окупності становить 0,76 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

ВИСНОВКИ

Під час виконання магістерської кваліфікаційної роботи було проаналізовано завдання, визначені завдання та поставлені вимоги, дотримавшись яких, можна досягти поставленої мети. Також були проаналізовані літературні джерела, які допоможуть у досягненні мети.

Були проаналізовані типи мережевих архітектур з точки зору ранговості. Після аналізу та порівняння було визначено, що пірингова архітектура є більш захищеною, оскільки дані, що надсилаються по мережі не зберігаються на третій стороні, а тільки на пристрої користувача. Такий підхід є безпечнішим, адже унеможливлює велику кількість атак, що пов’язані з центральним сервером.

Були проаналізовані типи піринговых мереж, їх історія та використання. Зокрема було виявлено, що пірингові мережі використовуються навіть на державному в Україні, а система «Трембіта» є однією з таких.

Також були проаналізовані атаки, які можуть бути реалізовані на пірингові мережі.

Було розроблено модель інформаційної технології, яка складається з шести процесів: процесу автентифікації, процесу створення ключів шифрування, процесу створення унікального ідентифікатора, процесу встановлення з’єднання, процесу зберігання даних та процесу обміну даними. Процеси взаємопов’язані і кожен наступний процес може використовувати дані, що були сформовані на попередньому. Для кожного з процесів було описано його математичну модель, а також математичну модель інформаційної технології вцілому.

На основі моделі інформаційної технології було розроблено архітектуру додатку, яка складається з чотирьох модулів: модулю автентифікації, модулю взаємодії з базою даних, криптографічного модулю, модулю обміну даними.

Після створення архітектури та на основі поставлених вимог, були розроблені алгоритми функціонування системи. Також, після порівняння декількох потокових шифрів, було обрано XChaCha20, оскільки він показав

найкращий результат у відношенні захищеності до швидкості серед усіх претендентів.

Після аналізу інструментів, які дозволяють створювати кросплатформні додатки, було обрано фреймворк Flutter, адже при порівнянні він показав найкращий результат.

Дотримуючись вимог, що були поставлені на етапі постановки задач, а також за допомогою розроблених архітектури та алгоритмів, було розроблено програмний застосунок для мобільних платформ, який в повному обсязі виконує поставлені задачі.

За допомогою додаткових інструментів було протестовано основні функції, які повинен виконувати програмний застосунок. Під час тестування помилок в роботі виявлено не було.

Було проведено комерційних аудит розробки, розраховано узагальнений коефіцієнт якості розробки, проаналізовані витрати, яких потребує розробка інформаційної технології. Після аналізу всіх витрати, було пораховано, що окупність розробленої системи становить ~0,76 року, що менше трьох років і це дає змогу зробити висновок, що розробка інформаційної технології є доцільною.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Distributed application architecture. Wayback Machine. URL: https://web.archive.org/web/20110406121920/http:/java.sun.com/developer/Books/jdbc/ch07.pdf (дата звернення: 05.10.2022).
2. Schollmeier R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. Institute of Communication Networks, Technische Universität München, 80333 München, Germany, 2002.
3. Johnson E., Kunze A. IXP1200 programming. Danvers : Copyright Clearence Center, 222, Rosewood drive, 2002. 275 c.
4. Saroiu S., Gummadi K., Gribble S. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. Seattle, 2002.
5. The tor project | privacy & freedom online. Tor Project | Anonymity Online. URL: https://www.torproject.org/about/history/ (дата звернення: 06.10.2022).
6. Ландэ Д. P2P - по секрету всему свету. Сети и бизнес. 2008. № 2. URL: http://dwl.kiev.ua /art/p2p/p2p-end.pdf (дата звернення: 06.10.2022).
7. About SETI@home. SETI@home. URL: https://setiathome.berkeley.edu /sah_about.php (дата звернення: 06.10.2022).
8. Smith R. Grid computing: a brief technology analysis. 2004.
9. Project RC5. distributed.net. URL: https://www.distributed.net/RC5 (дата звернення: 06.10.2022).
10. The Evolution of Peer-to-Peer File Sharing System. 360 Logica. URL: https://www.360logica.com/blog/the-evolution-of-peer-to-peer-file-sharing-system/ (дата звернення: 06.10.2022).
11. Кренцін М., Куперштейн Л. Аналіз тенденцій розвитку піринговых мереж. Вісник Хмельницького національного університету. Технічні науки. 2021. Т. 4, № 299.
12. Brain M. How Gnutella Works. HowStuffWorks. URL: https://computer.howstuffworks.com/file-sharing.htm (дата звернення: 07.10.2022).

13. Ghosemajumder S. Advanced peer-based technology business models. Massachusets, 2002.
14. Vellishetty T. Understanding BitTorrent protocol. BeautifulCode. 2019. URL: https://www.beautifulcode.co/blog/58-understanding-bittorrent-protocol (дата звернення: 07.10.2022).
15. The BitTorrent protocol specification. BitTorrent.org. URL: http://www.bittorrent.org/beps/bep_0003.html (дата звернення: 07.10.2022).
16. What is blockchain technology? - IBM blockchain | IBM. IBM - Deutschland | IBM. URL: https://www.ibm.com/topics/what-is-blockchain#:~:text=Blockchain%20overview,patents,%20copyrights,%20branding). (дата звернення: 08.10.2022).
17. What are smart contracts on blockchain? | IBM. IBM - Deutschland | IBM. URL: https://www.ibm.com/topics/smart-contracts (дата звернення: 08.10.2022).
18. Digital assets | finra.org. A vibrant market is at its best when it works for everyone. | FINRA.org. URL: https://www.finra.org/investors/investing/investment-products/digital-assets (дата звернення: 08.10.2022).
19. CeFi vs. defi — comparing centralized to decentralized finance / K. Qin et al. URL: https://arxiv.org/pdf/2106.08157.pdf (дата звернення: 08.10.2022).
20. Про систему. «ТРЕМБІТА» Система електронної взаємодії державних електронних інформаційних ресурсів. URL: https://trembita.gov.ua/ua/projects/trembita-vzayemodiya-reyestriv (дата звернення: 08.10.2022).
21. X-Road® technology overview — x-road® data exchange layer. X-Road® Data Exchange Layer. URL: https://x-road.global/x-road-technology-overview (дата звернення: 08.10.2022).
22. The pollution attack in P2P live video streaming:measurement results and defenses / P. Dhungel et al. Brooklyn. P. 323—328.
23. Mondal A., Kitsuregawa M. Privacy, security and trust in P2P environments: a perspective. 17th international conference on database and expert systems applications (DEXA'06), Krakow, Poland. URL: https://doi.org/10.1109/dexa.2006.116 (дата звернення: 08.10.2022).

24. Yang Y., Yang L. A survey of peer-to-peer attacks and counter attacks. Pomona. Pomona. URL: http://worldcomp-proceedings.com/proc/p2012/SAM9754.pdf (дата звернення: 08.10.2022).
25. Washbourne L. A survey of P2P network security. URL: https://arxiv.org/pdf/1504.01358.pdf (дата звернення: 08.10.2022).
26. Douceur J. R. The Sybil attack. ResearchGate. URL: https://www.researchgate.net/publication/221160492_The_Sybil_Attack (дата звернення: 09.10.2022).
27. Аналіз проблем безпеки піринговых мереж / Л. Куперштейн та ін. Інформаційні технології та комп'ютерна інженерія. 2022. Т. 54, № 2. С. 5—14. URL: https://doi.org/10.31649/1999-9941-2022-54-2-5-14 (дата звернення: 08.10.2022)
28. LDEPTH: A low diameter hierarchical P2P network architecture / N. Rahimi et al. 2016 IEEE 14th international conference on industrial informatics (INDIN), м. Poitiers, France, 19—21 лип. 2016 р. 2016. URL: https://doi.org/10.1109/indin.2016.7819275 (дата звернення: 27.11.2022).
29. Efficient data lookup in non-dht based low diameter structured P2P network / B. Gupta et al. м. Illinois. Carbondale. URL: https://www.cse.msstate.edu/wp-content/uploads/2020/04/I2_rahimi.pdf (дата звернення: 09.10.2022).
30. Saboori E., Mohammadi S. Anonymous communication in peer-to-peer networks for providing more privacy and security. Nternational journal of modeling and optimization. 2012. Vol. 2, no. 3.
31. Structural-based tunneling: preserving mutual anonymity forcircular P2P networks / A. Naghizadeh et al. International journal of communication systems. 2015. P. 602—619.
32. Chord: A scalable peer-to-peer lookup service for snternet applications / I. Stoica et al. 2002. URL: https://web.archive.org/web/20120722084813/http://pdos.csail.mit.edu/chord/papers/chord-tn.pdf (дата звернення: 28.10.2022).
33. The tox reference. Zetok | GitHub Pages. URL: https://zetok.github.io/tox-spec/ (дата звернення: 17.11.2022).

34. RFC 4122: A universally unique identifier (UUID) URN namespace / P. Leach et al. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc4122 (дата звернення: 17.11.2022).
35. Daemen J., Rijmen V. Note on naming. URL: https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines /documents /aes-development/rijndael-ammended.pdf#page=1 (дата звернення: 18.11.2022).
36. Biryukov A., Khovratovich D., Nikolić I. Distinguisher and Related-Key Attack on the Full AES-256. URL: https://link.springer.com/chapter/10.1007/978-3-642-03356-8_14 (дата звернення: 18.12.2022).
37. Sullivan N. Do the ChaCha: better mobile performance with cryptography. The Cloudflare Blog. URL: https://blog.cloudflare.com/do-the-chacha-better-mobile-performance-with-cryptography/ (дата звернення: 18.11.2022).
38. XChaCha20 encryption. Securely Store, Manage & Autofill Passwords | NordPass. URL: https://nordpass.com/features/xchacha20-encryption/#:~:text=NordPass%20uses%20the%20XChaCha20%20encryption,safe%20connections%20for%20their%20users. (дата звернення: 18.11.2022).
39. New features of latin dances: analysis of salsa, chacha, and rumba / J.-P. Aumasson et al. URL: https://eprint.iacr.org/2007/472.pdf (дата звернення: 18.11.2022).
40. Crowley P. Truncated differential cryptanalysis of five roundsof Salsa20. ciphergoth.org:. URL: http://www.ciphergoth.org/crypto/salsa20/ (дата звернення: 18.11.2022).
41. Lindell Y., Katz J. Introduction to modern cryptography. Taylor & Francis Group, 2014. 603 p.
42. Vanhoef M., Piessens F. Numerous occurrence monitoring & recovery exploit. RC4 NOMORE. URL: https://www.rc4nomore.com/ (дата звернення: 20.11.2022).
43. The eSTREAM portfolio page. ECRYPT. URL: https://www.ecrypt.eu.org/stream/ (дата звернення: 20.11.2022).
44. The stream cipher rabbit / M. Boesgaard et al. URL: https://www.ecrypt.eu.org /stream/p3ciphers/rabbit/rabbit_p3.pdf (дата звернення: 23.11.2022).

45. Lu Y., Wang H., Ling S. Cryptanalysis of Rabbit. Lecture Notes in Computer Science, м. Taipei, 18 груд. 2008 р. Taipei, 2008. P. 204—214.
46. Architectural overview of cordova platform - apache cordova. Apache Cordova. URL: https://cordova.apache.org/docs/en/latest/guide/overview/index.html (дата звернення: 10.12.2022).
47. Yaskiv A. Native mobile app development or react native vs flutter vs cordova: what to choose in 2021. Apiko | Learn. URL: https://apiko.com/blog/native-mobile-app-development-react-native-vs-flutter-vs-cordova/#3 (дата звернення: 10.12.2022).
48. Martin S. React native vs. flutter vs. ionic. Medium. URL: https://sophiamartin121.medium.com/react-native-vs-flutter-vs-ionic-46d3350f96ee (дата звернення: 11.12.2022).
49. Thomas G. What is flutter and why you should learn it in 2020. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/ (дата звернення: 11.12.2022).
50. Get | flutter package. Dart packages. URL: https://pub.dev/packages/get#about-get (дата звернення: 13.12.2022).
51. Flutter_nearby_connections | flutter package. Dart packages. URL: https://pub.dev/packages/flutter_nearby_connections (дата звернення: 13.12.2022).
52. Logcat command-line tool | Android Developers. Android Developers. URL: https://developer.android.com/studio/command-line/logcat (дата звернення: 18.12.2022).
53. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.
54. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

ДОДАТКИ

Додаток А. Результат перевірки роботи на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційна технологія захищеного обміну даними на основі піринговых мереж

Автор роботи: Притула Андрій Вікторович

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 98,33%. Схожість – 1,67%.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Каплун В. А.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Притула А. В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Кулаштейн І.Ц.
(підпис) (прізвище, ініціали)

Додаток Б. Лістинг програми

```
import 'package:flutter/material.dart';
import
'package:flutter_localizations/flutter_localiz
ations.dart';
import
'package:form_builder_validators/form_builder_
validators.dart';

import 'package:get/get.dart';
import
'package:peer_chat/app/data/secure_storage/pas
sword_storage.dart';
import
'package:peer_chat/app/data/secure_storage/sec
ure_storage_provider.dart';

import 'app/routes/app_pages.dart';

void main() async {
  runApp(
    GetMaterialApp(
      title: "Peer chat",
      initialRoute: AppPages.INITIAL,
      getPages: AppPages.routes,
      localizationsDelegates: const [

GlobalMaterialLocalizations.delegate,

GlobalWidgetsLocalizations.delegate,

FormBuilderLocalizations.delegate,
      ],
    ),
  );
}
import 'dart:convert';

import
'package:cryptography/cryptography.dart';
import
'package:peer_chat/app/data/entities/user.enti
ty.dart';
import
'package:peer_chat/app/services/hash.service.d
art';

class KeyService {
  final _algorithm = X25519();

  Future<SimpleKeyPair>
generateNewKeyPair() async {
    return                          await
_algorithm.newKeyPair();
  }

  Future<SimpleKeyPair>
generateNewKeyPairFromUserSeed(User      user)
async {
    return                          await
_algorithm.newKeyPairFromSeed(
      await HashService.hash(
        user.getSeed(),
      ),
    );
  }

  Future<PublicKey>
extractPublicKey(SimpleKeyPair pair) async {
    return                          await
pair.extractPublicKey();
  }

  Future<SecretKey> getSharedKey({
    required KeyPair keyPair,
    required PublicKey remotePublicKey,
  }) async {
    return                          await
_algorithm.sharedSecretKey(
        keyPair:               keyPair,
remotePublicKey: remotePublicKey);
  }
}
import 'dart:convert';

import
'package:cryptography/cryptography.dart';

class HashService {
  static final _hashAlg = Sha256();

  static                  Future<String>
base64Hash(String value) async {
    final      hashed      =      await
_hashAlg.hash(value.codeUnits);
    return base64Encode(hashed.bytes);
  }

  static              Future<List<int>>
hash(List<int> value) async {
```

```
            final      hashed      =      await
_hashAlg.hash(value);
            return hashed.bytes;
          }
        }
        import 'dart:convert';
        import 'dart:typed_data';

        import
'package:cryptography/cryptography.dart';

        class CryptoService {
          const CryptoService._();
          static     final     kAlgorithm     =
Xchacha20.poly1305Aead();

          static   const   kNonceLength   =   24,
kMacLength = 16;

          static              Future<Uint8List>
encryptConcatenation(
              SecretKey    secretKey,    String
plainText) async {
            final     encrypt     =     (await
kAlgorithm.encrypt(plainText.codeUnits,
secretKey: secretKey,))
                .concatenation();
            return encrypt;
          }

          static Future<SecretBox> encrypt(
              SecretKey    secretKey,    String
plainText) async {
            final     encrypt     =     (await
kAlgorithm.encrypt(plainText.codeUnits,
secretKey: secretKey,));
            return encrypt;
          }

          static Future<String> base64Encrypt(
              SecretKey    secretKey,    String
plainText) async {
            return           base64Encode(await
encryptConcatenation(secretKey, plainText));
          }

          static              Future<Uint8List>
decrypt(SecretKey  secretKey,  Uint8List  box)
async {
            final     decrypted     =     await
kAlgorithm.decrypt(
              SecretBox.fromConcatenation(
                box,
                nonceLength: kNonceLength,
                macLength: kMacLength,
              ),
              secretKey: secretKey,
            );

            return
Uint8List.fromList(decrypted);
          }

          static Future<String> base64Decrypt(
            SecretKey secretKey,
            String base64SecretBox,
          ) async {
            final     decrypted     =     await
decrypt(secretKey,
base64Decode(base64SecretBox));
            return utf8.decode(decrypted);
          }
        }
        import 'package:get/get.dart';

        import
'../modules/all_chats/bindings/all_chats.bindi
ng.dart';
        import
'../modules/all_chats/views/all_chats.view.dar
t';
        import
'../modules/auth/bindings/auth.binding.dart';
        import
'../modules/auth/views/auth.view.dart';
        import
'../modules/chat/bindings/chat.binding.dart';
        import
'../modules/chat/views/chat.view.dart';
        import
'../modules/home/bindings/home.binding.dart';
        import
'../modules/home/views/home.view.dart';
        import
'../modules/settings/bindings/settings.binding
.dart';
        import
'../modules/settings/views/settings.view.dart'
;
        import
'../modules/splash/bindings/splash.binding.dar
t';
        import
'../modules/splash/views/splash.view.dart';

        part 'app_routes.dart';
```

```
class AppPages {
  AppPages._();

  static const INITIAL = Routes.SPLASH;

  static final routes = [
    GetPage(
      name: _Paths.HOME,
      page: () => const HomeView(),
      binding: HomeBinding(),
    ),
    GetPage(
      name: _Paths.AUTH,
      page: () => const AuthView(),
      binding: AuthBinding(),
    ),
    GetPage(
      name: _Paths.SPLASH,
      page: () => const SplashView(),
      binding: SplashBinding(),
    ),
    GetPage(
      name: _Paths.CHAT,
      page: () => const ChatView(),
      binding: ChatBinding(),
    ),
    GetPage(
      name: _Paths.ALL_CHATS,
      page: () => const AllChatsView(),
      binding: AllChatsBinding(),
    ),
    GetPage(
      name: _Paths.SETTINGS,
      page: () => const SettingsView(),
      binding: SettingsBinding(),
    ),
  ];
}
part of 'app_pages.dart';
// DO NOT EDIT. This is code generated
via package:get_cli/get_cli.dart

abstract class Routes {
  Routes._();
  static const HOME = _Paths.HOME;
  static const AUTH = _Paths.AUTH;
  static const SPLASH = _Paths.SPLASH;
  static const CHAT = _Paths.CHAT;
  static     const     ALL_CHATS     =
_Paths.ALL_CHATS;
  static     const     SETTINGS     =
_Paths.SETTINGS;
}

abstract class _Paths {
  _Paths._();
  static const HOME = '/home';
  static const AUTH = '/auth';
  static const SPLASH = '/splash';
  static const CHAT = '/chat';
  static const ALL_CHATS = '/all-chats';
  static const SETTINGS = '/settings';
}
import 'package:get/get.dart';

import
'../controllers/home.controller.dart';

class HomeBinding extends Bindings {
  @override
  void dependencies() {
    Get.lazyPut<HomeController>(
      () => HomeController(),
    );
  }
}
import 'dart:async';
import 'dart:convert';

import
'package:cryptography/cryptography.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import
'package:flutter_nearby_connections/flutter_ne
arby_connections.dart';
import 'package:get/get.dart';
import
'package:peer_chat/app/data/database/daos/user
.dao.dart';
import
'package:peer_chat/app/data/entities/device_us
er.entity.dart';
import
'package:peer_chat/app/data/entities/message/d
ata_message.dart';
import
'package:peer_chat/app/data/entities/message/m
essage.entity.dart';
import
'package:peer_chat/app/data/entities/message/m
essage_type.enum.dart';
import
'package:peer_chat/app/data/entities/message/s
tart_message_data.dart';
```

```
import
'package:peer_chat/app/data/entities/user.enti
ty.dart';
import
'package:peer_chat/app/data/entities/user_min.
entity.dart';
import
'package:peer_chat/app/services/crypto.service
.dart';
import
'package:peer_chat/app/services/key.service.da
rt';
import 'package:uuid/uuid.dart';

class HomeController extends
GetxController {
  //TODO: Implement HomeController
  final usersDao = UserDao();
  final count = 0.obs;
  final nearbyService =
NearbyService();

  final devices =
<DeviceUserEntity>[].obs;

  late StreamSubscription
devicesSubscription;
  late StreamSubscription
dataSubscription;

  @override
  void onInit() {
    init();
    super.onInit();
  }

  Future<User> initDB() async {
    return user.value = await
usersDao.getUser();
  }

  late final user = Rxn<User>();

  init() async {
    final keyService = KeyService();

    /*print(
      keyService.getSharedKey(
          keyPair:
base64Decode(user.value!.keyPairData!),
          remotePublicKey:
SimplePublicKey(
            base64Decode(
'0I6+yrJtgNB5c3H0LO7kRs8CVbdHHw6NGyQ4Ss443EU='
,
            ),
            type: X25519().keyPairType,
          )),
    );*/
    await initDB();

    await nearbyService.init(
      serviceType: 'mpconn',
      deviceName: jsonEncode(
        user.value!.toUserMinEntity().toJson())
/*user.value!.fullName*/,
      strategy: Strategy.P2P_CLUSTER,
      callback: (isRunning) async {
        final encoded =
jsonEncode(user.value!.toJson());
        print(jsonDecode(encoded));
      },
    );

    devicesSubscription =
nearbyService.stateChangedSubscription(callbac
k: (devicesList) {
      devices.clear();

      devices.addAll(
        devicesList.map(
          (e) => DeviceUserEntity(
            device: e,
            user:
UserMinEntity.fromJson(
              jsonDecode(e.deviceName),
            ),
          ),
        ),
      );
    });

    dataSubscription =
nearbyService.dataReceivedSubscription(callbac
k: _messageReceived);
  }

  void _messageReceived(dynamic
receivedData) async {
    print("DL| ECRYPTED DATA RECEIVED:
" + receivedData['message']);
```

```
            final key = SecretKey(
base64Decode('MvP7BmHa9RgWFhLpSnivuBCvrpQOxr60
Vx/AVTReuBY='),
            );

            print("DL| KEY FOR DECRYPT DATA: "
+
'MvP7BmHa9RgWFhLpSnivuBCvrpQOxr60Vx/AVTReuBY='
);

            try {
              final wrongKey = SecretKey([
                50,
                243,
                251,
                6,
                97,
                218,
                245,
                24,
                22,
                22,
                18,
                233,
                74,
                120,
                175,
                184,
                16,
                175,
                174,
                148,
                14,
                198,
                190,
                180,
                87,
                31,
                192,
                85,
                52,
                94,
                184,
                21
              ]);
              print("DL| KEY FOR DECRYPT DATA:
" +
                  base64Encode(await
wrongKey.extractBytes()));
              final wrongDecryptedMessage =
                  await
CryptoService.base64Decrypt(wrongKey,
receivedData['message']);
              print("DL| WRONG DECRYPTED DATA:
" + wrongDecryptedMessage);
            } catch (e) {
              print("DL| ERROR: wrong
parameters for decryption");
            }

            final decryptedMessage =
                await
CryptoService.base64Decrypt(key,
receivedData['message']);

            print("DL| DECRYPTED DATA: " +
decryptedMessage);

            /* print("DL| DATA RECEIVED: " +
receivedData['message']);
            final KeyService keyService =
KeyService();
            final newKey = await
keyService.generateNewKeyPairFromUserSeed(user
.value!);
            final sharedKey = await
keyService.getSharedKey(
              keyPair: newKey,
              remotePublicKey: SimplePublicKey(
base64Decode(receivedData['message']),
                type: KeyPairType.x25519,
              ),
            );
            print('DL| SHARED KEY: ' +
base64Encode(await
sharedKey.extractBytes()));*/
            // print(receivedData);
            /*final data = receivedData as
MessageEntity;
            switch (data.type) {
              case MessageType.startConnection:
_sendPublicKeyToPeer(data.deviceId);
                break;
              case
MessageType.receivePublicKey:
_calculateSharedKey(data.startMessageData,
data.deviceId);
                break;
              case MessageType.dataMessage:
```

```
_handleDataMessage(data.dataMessage,
data.deviceId);
              break;
            default:
              throw    Exception('Unsupported
message type');
          }*/

          /*Get.snackbar("New       message",
data['message']);
          print(data);*/
        }

        void
_calculateSharedKey(StartMessageData?     data,
String? deviceId) {}

        void     _sendPublicKeyToPeer(String?
deviceId) {}

        void  _handleDataMessage(DataMessage?
data, String? deviceId) {}

        void startBrowsing() async {
          await
nearbyService.stopBrowsingForPeers();

nearbyService.startBrowsingForPeers();
        }

        void startAdvertising() async {
          await
nearbyService.stopAdvertisingPeer();

nearbyService.startAdvertisingPeer();
        }

        void connectToDevice(Device device) {
          nearbyService.invitePeer(
            deviceID: device.deviceId,
            deviceName: device.deviceName,
          );
        }

        void sendData(String deviceId, String
message) async {
          final encryptedMessageConcatenation
= await CryptoService.base64Encrypt(

SecretKey(base64Decode('MvP7BmHa9RgWFhLpSnivuB
CvrpQOxr60Vx/AVTReuBY=')),
            'Hello, Pavel',
          );

          print("DL| SEND ENCRYPTED DATA" +
encryptedMessageConcatenation);

          /*final  KeyService  keyService  =
KeyService();
          final      newKey      =      await
keyService.generateNewKeyPairFromUserSeed(user
.value!);
          final     pkBytes     =     (await
newKey.extractPublicKey()).bytes;
          print('DL|   SEND   DATA:   '   +
base64Encode(pkBytes));*/
          nearbyService.sendMessage(deviceId,
encryptedMessageConcatenation);
          //
nearbyService.sendMessage(deviceId,
user.value!.publicKey!);
        }

        Future<void> _saveMessageToDatabase(
            String      deviceId,      String
encryptedMessage) async {}

        @override
        void onReady() {
          super.onReady();
        }

        @override
        void onClose() {
          super.onClose();
        }

        Future<void>
showUserBottomsheet(String id) async {
          Get.bottomSheet(
            ListView(
              shrinkWrap: true,
              children: [
                ListTile(
                  title:        Text('Approve
user'),
                ),
                ListTile(
                  title:   Text('Check   user
ID'),
                  onTap: () => seeUserId(id),
                ),
              ],
            ),
            backgroundColor: Colors.white
```

```
          );
        }

        Future<void> seeUserId(String id)
async {
          Get.defaultDialog(
            title: "Remote User id",
            content: Column(
              children: [
                Padding(
                  padding:              const
EdgeInsets.symmetric(vertical: 8.0),
                  child:   SelectableText(id,
style: const TextStyle(fontSize: 13)),
                ),
                IconButton(
                  onPressed: () {
                    Clipboard.setData(
                      ClipboardData(text:
id),
                    );
                  },
                  icon:                 const
Icon(Icons.content_copy),
                ),
              ],
            ),
          );
        }
      }

      /*void keys() async {
          final keyService = KeyService();
          // Alice chooses her key pair
          final    aliceKeyPair    =    await
keyService.generateNewKeyPairFromUserSeed(
            const User(
              username: 'alice',
              password: 'alicepassword',
            ),
          );
          final   alicePublicKey   =   await
aliceKeyPair.extractPublicKey();
          // Alice knows Bob's public key
          final    bobKeyPair    =    await
keyService.generateNewKeyPairFromUserSeed(
            const User(
              username: 'bob',
              password: 'bobpassword',
            ),
          );
          final    bobPublicKey    =    await
bobKeyPair.extractPublicKey();
          // Alice calculates the shared
secret.
          final  sharedAliceSecret  =  await
keyService.getSharedKey(
            keyPair: aliceKeyPair,
            remotePublicKey: bobPublicKey,
          );
          // Bob calculates the shared secret.
          final   sharedBobSecret   =   await
keyService.getSharedKey(
            keyPair: bobKeyPair,
            remotePublicKey: alicePublicKey,
          );
          print("==== ALICE KEYS ====");
          final alicePrivateKeyBytes = await
aliceKeyPair.extractPrivateKeyBytes();
          print("\tPRIVATE KEY");
          print("\t\t$alicePrivateKeyBytes");

print("\t\t${base64Encode(alicePrivateKeyBytes
)}");
          print("\tPUBLIC KEY");

print("\t\t${alicePublicKey.bytes}");

print("\t\t${base64Encode(alicePublicKey.bytes
)}");
          print('\n');
          print("==== BOB KEYS ====");
          final  bobPrivateKeyBytes  =  await
bobKeyPair.extractPrivateKeyBytes();
          print("\tPRIVATE KEY");
          print("\t\t${bobPrivateKeyBytes}");

print("\t\t${base64Encode(bobPrivateKeyBytes)}
");
          print("\tPUBLIC KEY");
          print("\t\t${bobPublicKey.bytes}");

print("\t\t${base64Encode(bobPublicKey.bytes)}
");
          print('\n');
          print("==== SHARED KEYS ====");
          final  aliceSharedKeyBytes  =  await
sharedAliceSecret.extractBytes();
          final  bobSharedKeyBytes  =  await
sharedBobSecret.extractBytes();
          print("\tALICE SHARED KEY");
          print("\t\t$aliceSharedKeyBytes");

print("\t\t${base64Encode(aliceSharedKeyBytes)
}");
          print("\tBOB SHARED KEY");
```

```
            print("\t\t$bobSharedKeyBytes");

print("\t\t${base64Encode(bobSharedKeyBytes)}"
);
            print(
                "==== IS KEYS EQUAL: ${const
ListEquality().equals(aliceSharedKeyBytes,
bobSharedKeyBytes)} ====");
            print('\n');
            final     encrypted     =     await
encrypt(sharedAliceSecret);
            decrypt(sharedBobSecret,
encrypted);
          }
          Future<String>      encrypt(SecretKey
secretKey) async {
            final     encrypted     =     await
CryptoService.base64Encrypt(
              secretKey,
              '''hello''',
            );
            print(encrypted);
            return encrypted;
          }
          Future<void>        decrypt(SecretKey
secretKey, String box) async {
            final     decrypted     =     await
CryptoService.base64Decrypt(
              secretKey,
              box,
            );
          }*/
        import
'package:cryptography/cryptography.dart';
        import 'package:flutter/material.dart';

        import 'package:get/get.dart';
        import
'package:peer_chat/app/core/widgets/drawer.dar
t';

        import
'../controllers/home.controller.dart';

        class          HomeView          extends
GetView<HomeController> {
          const    HomeView({Key?    key})    :
super(key: key);

          @override
          Widget build(BuildContext context) {
            return DefaultTabController(
              length: 2,
              child: Scaffold(
                drawer: Obx(() {
                  if (controller.user.value ==
null) return const SizedBox();
                  return        AppDrawer(user:
controller.user.value!);
                }),
                appBar: AppBar(
                  title:     const     Text('All
Chats'),
                  centerTitle: true,
                  bottom: TabBar(
                    tabs: [
                      Tab(
                        icon:
Icon(Icons.add_ic_call_outlined),
                        text: 'All',
                      ),
                      Tab(
                        icon:
Icon(Icons.done_all),
                        text: 'Approved',
                      ),
                    ],
                  ),
                ),
                body: TabBarView(
                  children: [
                    _buildAll(),
                    _buildAll(),
                  ],
                ),
              ),
            );
          }

          Widget _buildAll() {
            return Center(
              child: Column(
                children: [
                  /*OutlinedButton(
                    onPressed:
controller.startBrowsing,
                    child:                 const
Text('Browse'),
                  ),
                  OutlinedButton(
                    onPressed:
controller.startAdvertising,
                    child:                 const
Text('Advertise'),
                  ),*/
                  ObxValue(
```

```
                    (devices)                =>
ListView.separated(
                      shrinkWrap: true,
                      itemBuilder: (_, index) {
                        final      device      =
devices[index];
                        return ListTile(
                          title:
Text(device.user.fullName),
                          onLongPress:   ()   =>
controller.showUserBottomsheet(device.user.uui
d),
                          leading: IconButton(
                            icon:          const
Icon(Icons.send),
                            onPressed:   ()   =>
controller.sendData(

device.device.deviceId,
                              'message',
                            ),
                          ),
                          /*trailing:
IconButton(
                            icon:          const
Icon(Icons.connect_without_contact),
                            onPressed:   ()   =>
controller.connectToDevice(device.device),
                          ),*/
                        );
                      },
                      separatorBuilder: (_, __)
=> Container(
                        width: 1,
                        height: 1,
                        color: Colors.black,
                      ),
                      itemCount:
devices.value.length,
                    ),
                    controller.devices,
                  ),
                ],
              ),
            );
          }
        }
        import 'package:flutter/material.dart';
        import 'package:get/get.dart';
        import
'package:peer_chat/app/core/widgets/user_avata
r.dart';
        import
'package:peer_chat/app/data/entities/user.enti
ty.dart';

        import '../../routes/app_pages.dart';

        class AppDrawer extends StatefulWidget
{
          const AppDrawer({
            Key? key,
            required this.user,
          }) : super(key: key);

          final User user;

          @override
          State<AppDrawer>   createState()   =>
_AppDrawerState();
        }

        class      _AppDrawerState      extends
State<AppDrawer> {
          @override
          Widget build(BuildContext context) {
            return FractionallySizedBox(
              widthFactor: 0.7,
              child: Container(
                color: Colors.white,
                child: ListView(
                  //   Important:   Remove   any
padding from the ListView.
                  padding: EdgeInsets.zero,
                  children: [
                    DrawerHeader(
                      child: Column(
                        children: [
                          UserAvatar(
                            user: widget.user,
                            radius: 30,
                            fontSize: 20,
                          ),
                          const
SizedBox(height: 30,),
                          Text(

widget.user.fullName,
                            style:        const
TextStyle(fontSize: 25),
                          ),
                        ],
                      ),
                    ),
                    ListTile(
```

```
                    title:  const  Text('All
chats'),
                    leading:
Icon(Icons.all_inbox),
                    onTap: () {

Get.toNamed(Routes.ALL_CHATS);
                    },
                  ),
                  ListTile(
                    title:             const
Text('Home'),
                    leading:
Icon(Icons.all_inbox),
                    onTap: () {

Get.toNamed(Routes.HOME);
                    },
                  ),
                  ListTile(
                    title:             const
Text('Settings'),
                    leading:
Icon(Icons.settings),
                    onTap: () {

Get.toNamed(Routes.SETTINGS);
                    },
                  ),
                ],
              ),
            ),
          );
        }
      }
      import 'package:flutter/material.dart';
      import 'package:path/path.dart';
      import
'package:peer_chat/app/data/entities/user.enti
ty.dart';

      class       UserAvatar       extends
StatelessWidget {
        final User user;
        const UserAvatar({
          Key? key,
          required this.user,
          this.fontSize,
          this.radius = 16,
        }) : super(key: key);

        final double? fontSize, radius;

        String getInitials(User user) {
          final          splitted          =
user.fullName.split(' ');
          return       splitted[0][0]       +
splitted[1][0];
        }

        @override
        Widget build(BuildContext context) {
          return Container(
            margin:                      const
EdgeInsets.only(right: 8),
            child: GestureDetector(
              child: CircleAvatar(
                backgroundColor:        const
Color(0xfff5a2d9),
                radius: radius,
                child: Text(
                  getInitials(user),
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: fontSize ?? 12,
                    fontWeight:
FontWeight.w800,
                    height: 1.333,
                  ),
                ),
              ),
            ),
          );
        }
      }
      import 'dart:async';

      import 'package:get/get.dart';
      import
'package:get/get_core/src/get_main.dart';
      import 'package:path/path.dart';
      import
'package:path_provider/path_provider.dart';
      import
'package:peer_chat/app/data/secure_storage/pas
sword_storage.dart';
      import 'package:sembast/sembast.dart';
      import
'package:sembast/sembast_io.dart';

      import 'encrypt_codec.dart';

      class AppDatabase {

        // Singleton instance
```

```
        static final AppDatabase _singleton =
AppDatabase._();

        // Singleton accessor
        static AppDatabase get instance =>
_singleton;

        // Completer is used for transforming
synchronous code into asynchronous code.
        Completer<Database>?
_dbOpenCompleter;

        // A private constructor. Allows us to
create instances of AppDatabase
        // only from within the AppDatabase
class itself.
        AppDatabase._();

        Database? _database;

        // Database object accessor
        Future<Database> get database async {
          //    If    completer    is    null,
AppDatabaseClass is newly instantiated, so
database is not yet opened
          if (_dbOpenCompleter == null) {
            _dbOpenCompleter = Completer();
            // Calling _openDatabase will also
complete the completer with database instance
            _openDatabase();
          }
          //  If  the  database  is  already
opened,  awaiting  the  future  will  happen
instantly.
          // Otherwise, awaiting the returned
future will take some time - until complete()
is called
          //    on    the    Completer    in
_openDatabase() below.
          return _dbOpenCompleter!.future;
        }

        Future _openDatabase() async {
          final       passwordStorage       =
Get.find<PasswordStorage>();

          var             codec             =
getEncryptSembastCodec(password:         (await
passwordStorage.readPassword())!);
          // Get a platform-specific directory
where persistent app data can be stored
          final    appDocumentDir    =    await
getApplicationDocumentsDirectory();
          print(appDocumentDir);
          // Path with the form: /platform-
specific-directory/chat.db
          final           dbPath           =
join(appDocumentDir.path, 'chat.db');

          final     database     =     await
databaseFactoryIo.openDatabase(dbPath,   codec:
codec);
          // Any code awaiting the Completer's
future will now start executing

_dbOpenCompleter!.complete(database);
        }

        Future<Database>        create(String
password) async {
          _dbOpenCompleter ??= Completer();
          var             codec             =
getEncryptSembastCodec(password: password);
          // Get a platform-specific directory
where persistent app data can be stored
          final    appDocumentDir    =    await
getApplicationDocumentsDirectory();
          // Path with the form: /platform-
specific-directory/chat.db
          final           dbPath           =
join(appDocumentDir.path, 'chat.db');

          final     database     =     await
databaseFactoryIo.openDatabase(dbPath,   codec:
codec);
          // Any code awaiting the Completer's
future will now start executing

_dbOpenCompleter!.complete(database);
          return _dbOpenCompleter!.future;
        }
      }
      import 'dart:convert';
      import 'dart:math';
      import 'dart:typed_data';

      import 'package:crypto/crypto.dart';
      import 'package:encrypt/encrypt.dart';
      import 'package:sembast/sembast.dart';

      var _random = Random.secure();

      /// Random bytes generator
      Uint8List _randBytes(int length) {
        return Uint8List.fromList(
```

```
            List<int>.generate(length, (i) =>
_random.nextInt(256)));
        }

        /// Generate an encryption password
based on a user input password
        ///
        /// It uses MD5 which generates a 16
bytes blob, size needed for Salsa20
        Uint8List
_generateEncryptPassword(String password) {
          var              blob              =
Uint8List.fromList(md5.convert(utf8.encode(pas
sword)).bytes);
          assert(blob.length == 16);
          return blob;
        }

        /// Salsa20 based encoder
        class      _EncryptEncoder      extends
Converter<dynamic, String> {
          final Salsa20 salsa20;

          _EncryptEncoder(this.salsa20);

          @override
          String convert(dynamic input) {
            // Generate random initial value
            final iv = _randBytes(8);
            final          ivEncoded          =
base64.encode(iv);
            assert(ivEncoded.length == 12);

            // Encode the input value
            final encoded =

Encrypter(salsa20).encrypt(json.encode(input),
iv: IV(iv)).base64;

            // Prepend the initial value
            return '$ivEncoded$encoded';
          }
        }

        /// Salsa20 based decoder
        class      _EncryptDecoder      extends
Converter<String, dynamic> {
          final Salsa20 salsa20;

          _EncryptDecoder(this.salsa20);

          @override
          dynamic convert(String input) {
            // Read the initial value that was
prepended
            assert(input.length >= 12);
            final              iv              =
base64.decode(input.substring(0, 12));

            // Extract the real input
            input = input.substring(12);

            // Decode the input
            var            decoded            =
json.decode(Encrypter(salsa20).decrypt64(input
, iv: IV(iv)));
            if (decoded is Map) {
              return       decoded.cast<String,
dynamic>();
            }
            return decoded;
          }
        }

        /// Salsa20 based Codec
        class       _EncryptCodec       extends
Codec<dynamic, String> {
          late _EncryptEncoder _encoder;
          late _EncryptDecoder _decoder;

          _EncryptCodec(Uint8List
passwordBytes) {
            var            salsa20            =
Salsa20(Key(passwordBytes));
            _encoder                          =
_EncryptEncoder(salsa20);
            _decoder                          =
_EncryptDecoder(salsa20);
          }

          @override
          Converter<String,     dynamic>     get
decoder => _decoder;

          @override
          Converter<dynamic,     String>     get
encoder => _encoder;
        }

        /// Our plain text signature
        const      _encryptCodecSignature      =
'encrypt';

        /// Create a codec to use to open a
database with encrypted stored data.
        ///
```

```
        /// Hash (md5) of the password is used
(but never stored) as a key to encrypt
        /// the data using the Salsa20 algorithm
with a random (8 bytes) initial value
        ///
        /// This is just used as a demonstration
and should not be considered as a
        /// reference since its implementation
(and storage format) might change.
        ///
        /// No performance metrics has been made
to check whether this is a viable
        /// solution for big databases.
        ///
        /// The usage is then
        ///
        /// ```dart
        /// // Initialize the encryption codec
with a user password
        ///         var         codec         =
getEncryptSembastCodec(password:
'[your_user_password]');
        /// // Open the database with the codec
        ///     Database     db     =     await
factory.openDatabase(dbPath, codec: codec);
        ///
        /// // ...your database is ready to use
        /// ```
        SembastCodec
getEncryptSembastCodec({required        String
password}) => SembastCodec(
              signature:
_encryptCodecSignature,
              codec: _EncryptCodec(

_generateEncryptPassword(password),
              ),
            );
        import 'package:get/get.dart';
        import
'package:peer_chat/app/data/entities/user.enti
ty.dart';
        import 'package:sembast/sembast.dart';

        class UserDao {
          static   const   String   kUsersDB   =
'users';

          final           _usersDB           =
intMapStoreFactory.store(kUsersDB);

          Future<Database> get _db async {
            return Get.find<Database>();
          }

          Future insert(User user) async {
            await    _usersDB.add(await    _db,
user.toJson());
          }

          Future<User> getUser() async {
            // Finder object can also sort data.
            /*final finder = Finder(sortOrders:
[
              SortOrder('id'),
            ]);*/

            final   recordSnapshots   =   await
_usersDB.find(
              await _db,
            );

            //  Making  a  List<Fruit>  out  of
List<RecordSnapshot>
            return recordSnapshots
                .map((snapshot) {
                  return
User.fromJson(snapshot.value);
                })
                .toList()
                .first;
          }
        }
        import 'package:get/get.dart';
        import
'package:peer_chat/app/data/entities/user.enti
ty.dart';
        import 'package:sembast/sembast.dart';

        class UserDao {
          static   const   String   kUsersDB   =
'users';

          final           _usersDB           =
intMapStoreFactory.store(kUsersDB);

          Future<Database> get _db async {
            return Get.find<Database>();
          }

          Future insert(User user) async {
            await    _usersDB.add(await    _db,
user.toJson());
          }

          Future<User> getUser() async {
```

```
            // Finder object can also sort data.
            /*final finder = Finder(sortOrders:
[
              SortOrder('id'),
            ]);*/

            final   recordSnapshots   =   await
_usersDB.find(
              await _db,
            );

            //  Making  a  List<Fruit>  out  of
List<RecordSnapshot>
            return recordSnapshots
                .map((snapshot) {
                  return
User.fromJson(snapshot.value);
                })
                .toList()
                .first;
          }
        }
        import
'package:flutter_nearby_connections/flutter_ne
arby_connections.dart';
        import
'package:freezed_annotation/freezed_annotation
.dart';
        import
'package:flutter/foundation.dart';
        import
'package:peer_chat/app/data/entities/user_min.
entity.dart';

        import 'user.entity.dart';

        part 'device_user.entity.freezed.dart';
        part 'device_user.entity.g.dart';

        @freezed
        class       DeviceUserEntity       with
_$DeviceUserEntity {
          const factory DeviceUserEntity({
            @JsonKey(fromJson:  deviceFromJson,
toJson: deviceToJson) required Device device,
            required UserMinEntity user,
          }) = _DeviceUserEntity;

          factory
DeviceUserEntity.fromJson(Map<String,  dynamic>
json) =>
              _$DeviceUserEntityFromJson(json);

          //Map<String,  dynamic>  toJson()  =>
_$DeviceUserEntityToJson(this);
        }

        Device       deviceFromJson(Map<String,
dynamic> json) {
          return       Device(json['deviceId'],
json['deviceName'], json['state']);
        }

        Map<String,                       dynamic>
deviceToJson(Device d) {
          return {};
        }
        import 'dart:convert';

        import
'package:freezed_annotation/freezed_annotation
.dart';
        import
'package:flutter/foundation.dart';
        import
'package:peer_chat/app/data/entities/user_min.
entity.dart';
        import
'package:peer_chat/app/services/hash.service.d
art';



        part 'user.entity.freezed.dart';
        part 'user.entity.g.dart';

        @freezed
        class User with _$User {
          const User._();

          const factory User({
            required String id,
            required String fullName,
            required String username,
            required String password,
            String? keyPairData,
            String? uuid,
            @Default(false) bool isConnected,
          }) = _User;

          factory     User.fromJson(Map<String,
dynamic> json) => _$UserFromJson(json);

          List<int> getSeed() {
            return      [...username.codeUnits,
...password.codeUnits];
          }
```

```
          Future<String>    getHashedPassword()
async {
            return                          await
HashService.base64Hash(password);
          }

          UserMinEntity toUserMinEntity() {
            return      UserMinEntity(fullName:
fullName, uuid: id,);
          }
        }
        import
'package:freezed_annotation/freezed_annotation
.dart';
        import
'package:flutter/foundation.dart';

        part 'user_min.entity.freezed.dart';
        part 'user_min.entity.g.dart';

        @freezed
        class         UserMinEntity         with
_$UserMinEntity{

          const factory UserMinEntity({
            required String fullName,
            required String uuid,
          }) = _UserMinEntity;

          factory
UserMinEntity.fromJson(Map<String,     dynamic>
json) =>
              _$UserMinEntityFromJson(json);

          //Map<String,  dynamic>  toJson()  =>
_$UserMinEntityToJson(this);
        }
```

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ЗАХИЩЕНОГО ОБМІНУ ДАНИМИ НА ОСНОВІ ПІРИНГОВИХ МЕРЕЖ

СКЛАДОВІ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

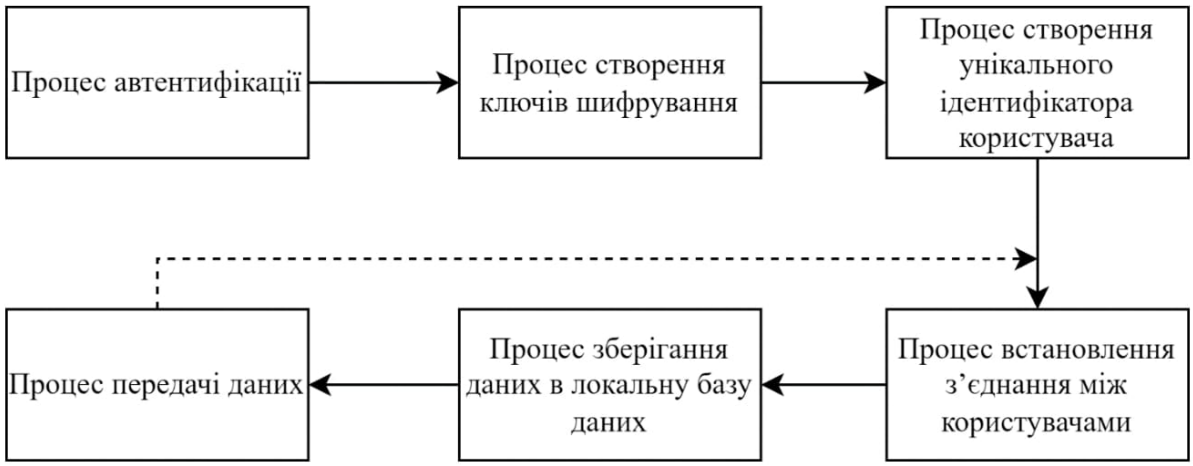


СХЕМА ПРОЦЕСУ АВТЕНТИФІКАЦІЇ

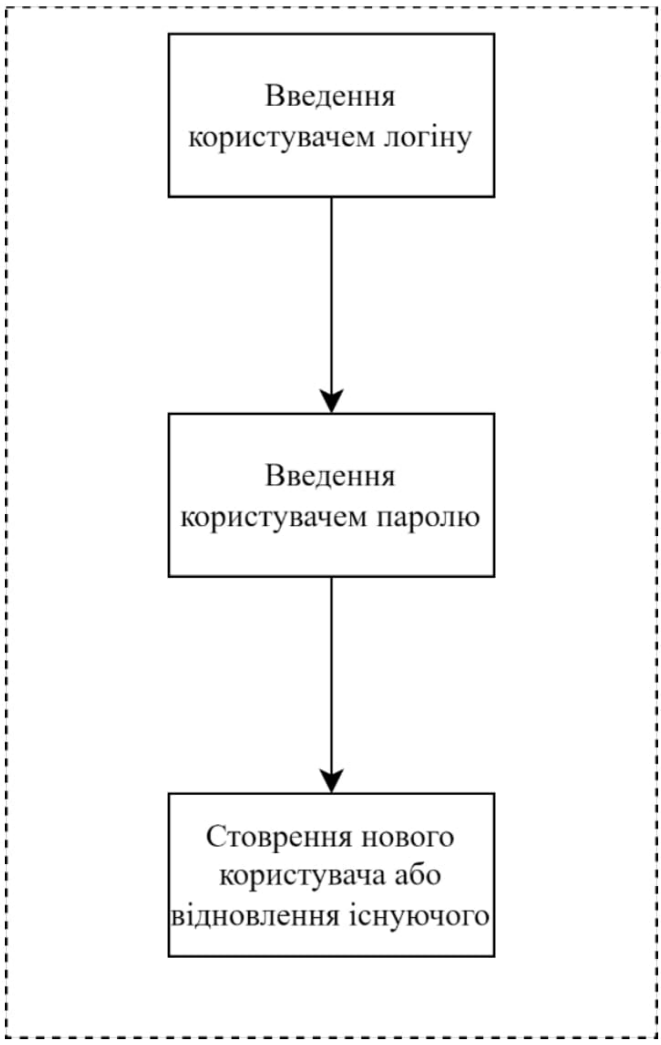


СХЕМА ПРОЦЕСУ СТВОРЕННЯ КЛЮЧІВ ШИФРУВАННЯ

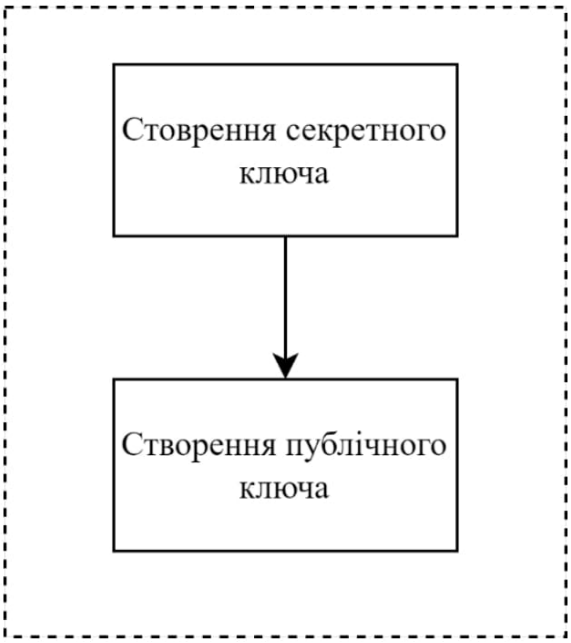


СХЕМА ПРОЦЕСУ СТВОРЕННЯ УНІКАЛЬНОГО ІДЕНТИФІКАТОРА

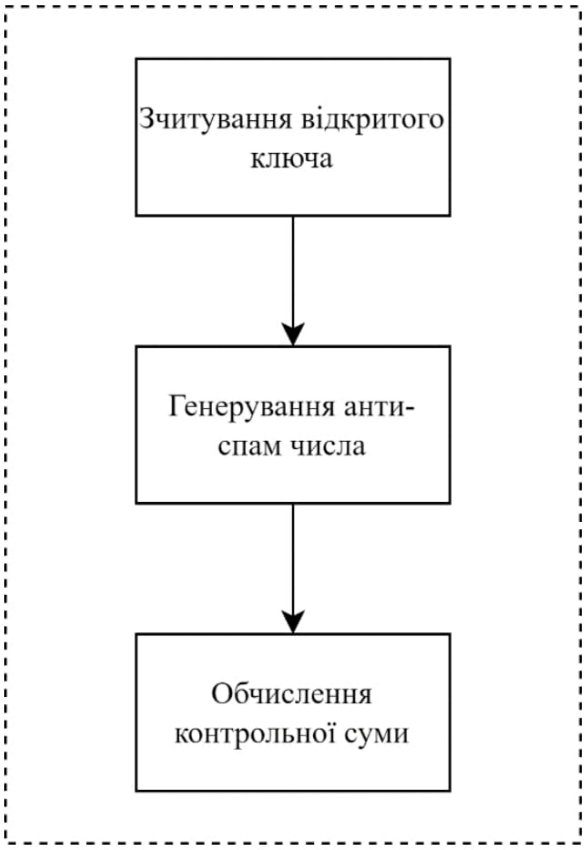
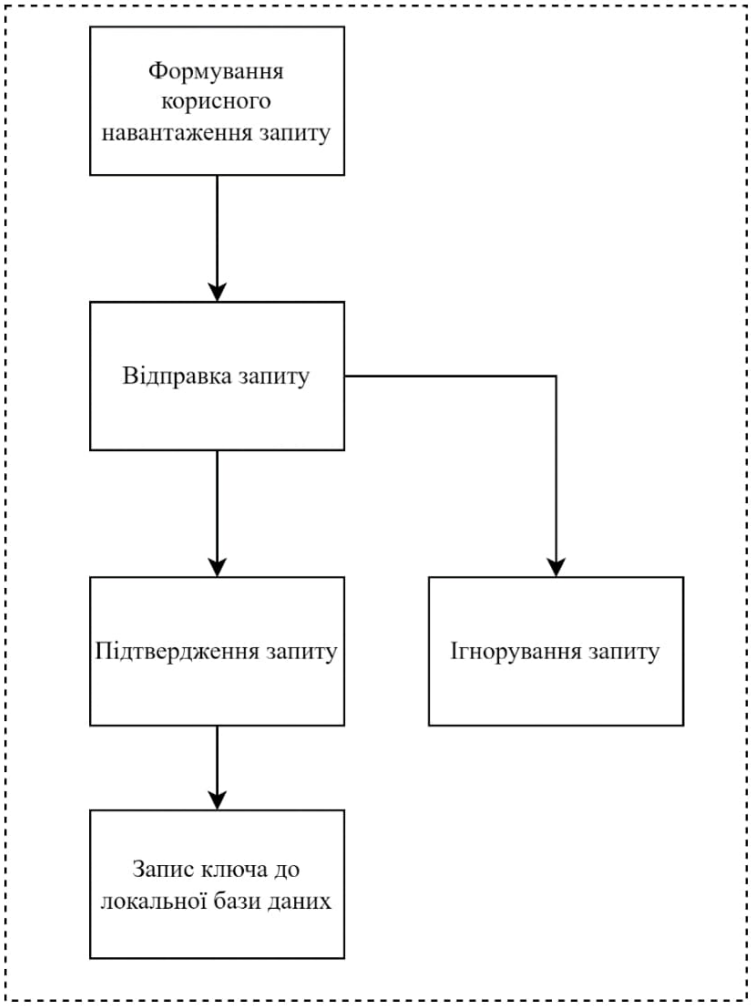
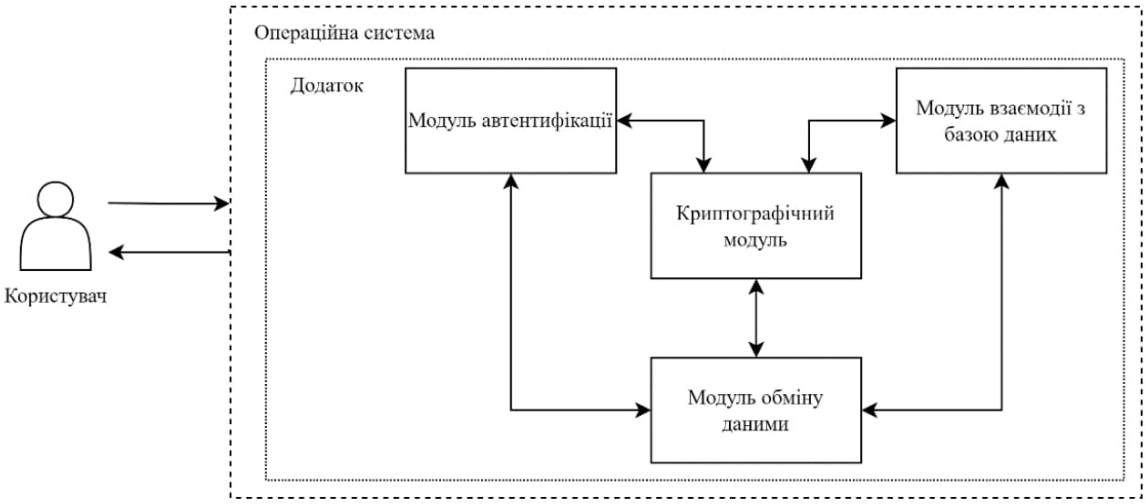


СХЕМА ПЕРЕДАЧІ ДАНИХ

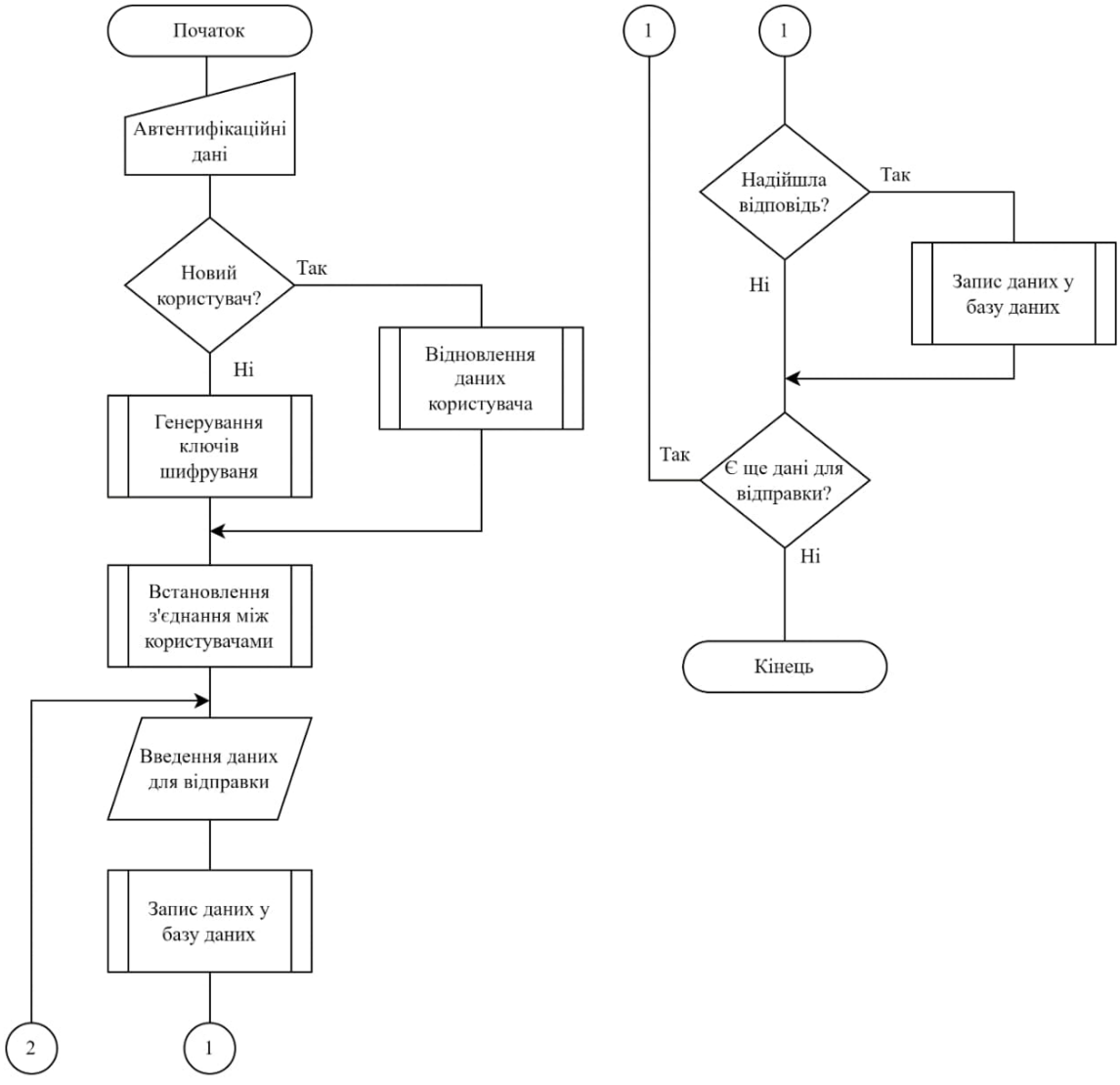
СХЕМА ПРОЦЕСУ ВСТАНОВЛЕННЯ З’ЄДНАННЯ МІЖ КОРИСТУВАЧАМИ



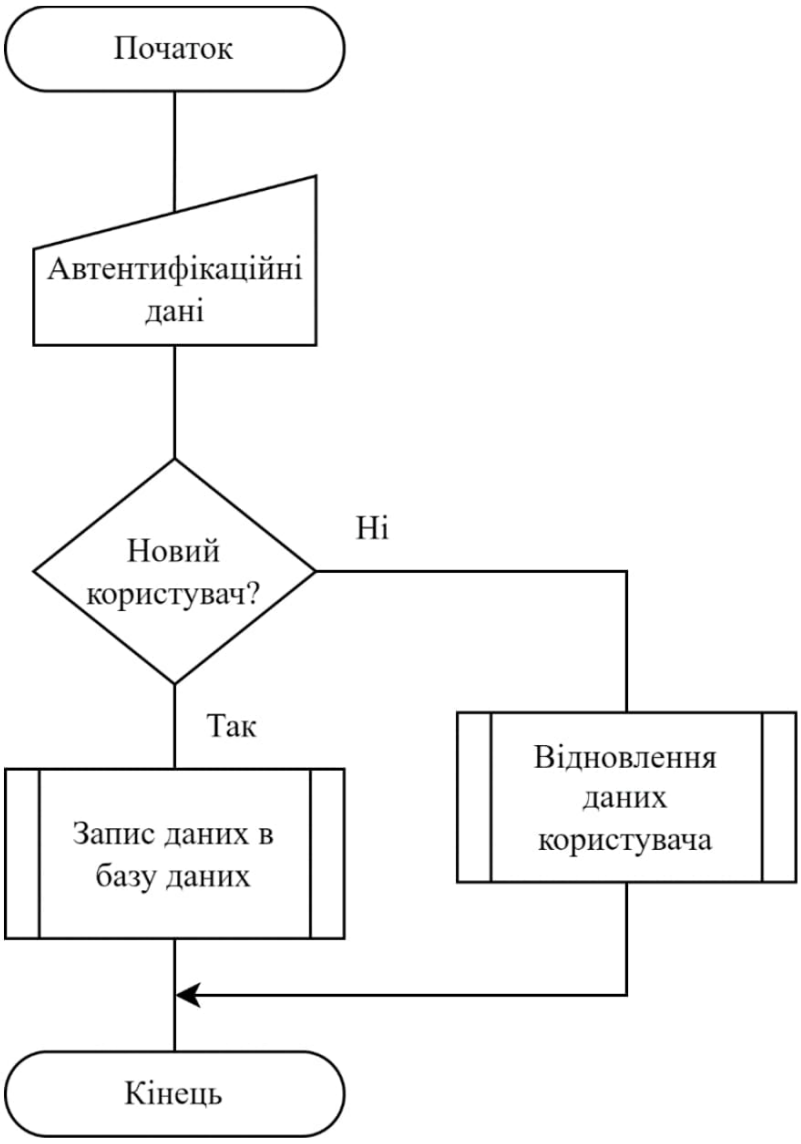
ЗАГАЛЬНА АРХІТЕКТУРА СИСТЕМИ



ЗАГАЛЬНИЙ АЛГОРИТМ РОБОТИ ДОДАТКУ



ЗАГАЛЬНИЙ АЛГОРИТМ РОБОТИ МОДУЛЮ АВТЕНТИФІКАЦІЇ



ЗАГАЛЬНИЙ АЛГОРИТМ ГЕНЕРУВАННЯ КЛЮЧІВ ШИФРУВАННЯ

ЗАГАЛЬНА СХЕМА ОБМІНУ КЛЮЧАМИ ШИФРУВАННЯ

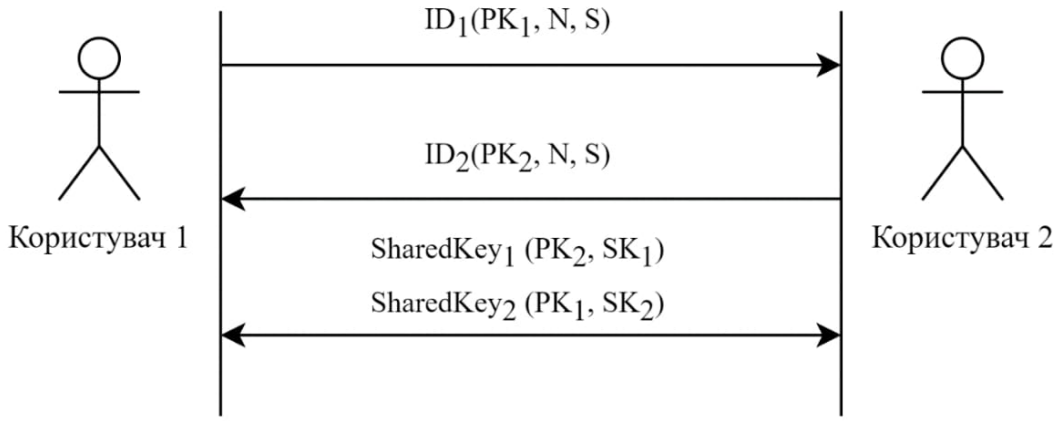


СХЕМА ПЕРЕДАЧІ ДАНИХ ПО МЕРЕЖІ