Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка месенджера для конфеденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, технологій стеганографії та моделей штучного інтелекту для маскування прихованих даних

Виконав: студент 4 курсу, гр.2КІТС-21б
спеціальності 125 – Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Жолукевський В.В._____
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. МБІС
Грицак А.В._____
(прізвище та ініціали)
«_3_» _червня_ 2025 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Крупельницький Л.В._____
(прізвище та ініціали)
«_3_» _червня_ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“_3_” _червня_ 2025р.

Вінниця ВНТУ – 2025

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

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

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

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

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Жолукевський Владислав Володимирович
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка месенджера для конфеденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, техологій стеганографії та моделей штучного інтелекту для маскування прихованих даних
Керівник роботи Грицак Анатолій Васильович к.т.н., доц., каф. МБІС
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 20 " березня 2025 року № 97
2. Термін подання студентом роботи 29 травня 2025 року
3. Вихідні дані до роботи Метою роботи є розробка месенджера для конфіденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, технологій стеганографії та моделей штучного інтелекту для маскування прихованих даних
4. Зміст текстової частини: Аналіз роботи алгоритмів шифрування AES та RSA, аналіз моделі штучного інтелекту, аналіз стеганографічного алгоритму StegCloak, розробка застосунку, реалізація застосунку.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
Слайд з зображенням стеганографічного алгоритму StagCloak, зображення схеми роботи криптографічного алгоритму AES та RSA

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Грицак А.В доц. кафедри МБІС		
Розділ II	Грицак А.В доц. кафедри МБІС		
Розділ III	Грицак А.В доц. кафедри МБІС		

7.Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напряму роботи	20.03.2025	24.03.2025	виконано
2	Аналіз предметної області	25.03.2025	04.04.2025	виконано
3	Розробка алгоритму системи	05.04.2025	09.04.2025	виконано
4	Проектування архітектури додатку	10.04.2025	19.04.2025	виконано
5	Реалізація модулів шифрування	20.04.2025	23.04.2025	виконано
6	Тестування системи	24.04.2025	28.04.2025	виконано
7	Оцінка результатів	29.04.2025	30.04.2025	виконано
8	Підготовка висновків	01.05.2025	02.05.2025	виконано
9	Оформлення звітів	03.05.2025	24.05.2025	виконано
10	Завершення оформлення	25.05.2025	28.05.2025	виконано

Студент _____ Жолукевський В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Грицак А.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У даній бакалаврській дипломній роботі проаналізовано та здійснено програмну реалізацію месенджеру для конфеденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, технологій стеганографії та моделей штучного інтелекту для маскування прихованих даних.

Виконано аналіз існуючих рішень для обмін інформацією в текстовому вигляді між двома співрозмовниками. Детально розглянуто процес наскрізного шифрування, роботи SSL для обміну даними між клієнтами та клієнтом і сервером а також процес приховування текстових даних в тексті. Описано архітектуру, інфраструктуру додатку та бази даних. Обґрунтовано вибір мови програмування, технологій та побудови інфраструктури а також описано удосконалений процес обміну даними з використанням наскрізного шифрування. Наведено приклад роботи застосунку.

Ключові слова: стеганографія, наскрізне шифрування, месенджер, SSL, веб-додаток, штучний інтелект.

ANNOTATION

This bachelor's thesis analyzes and implements a software implementation of a messenger for confidential information exchange using end-to-end encryption based on AES and RSA algorithms, steganography technologies and artificial intelligence models for masking hidden data.

An analysis of existing solutions for exchanging information in text form between two interlocutors is performed. The process of end-to-end encryption, SSL operation for data exchange between clients and client and server, as well as the process of hiding text data in the text are considered in detail. The architecture, infrastructure of the application and database are described. The choice of programming language, technologies and infrastructure construction are justified, and an improved process of data exchange using end-to-end encryption is described. An example of the application is given.

Keywords: steganography, end-to-end encryption, messenger, SSL, web application, artificial intelligence.

ЗМІСТ

ВСТУП 7
1 ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ТЕХНОЛОГІЙ 9
1.1 Аналіз алгоритмів AES та RSA 9
1.2 Концепція наскрізного шифрування 13
1.3 Приховування секрету в строкових повідомленнях 15
1.4 Штучний інтелект та його застосування для маскування даних 18
1.5 Висновки 21
2 ПРОЕКТУВАННЯ МЕСЕНДЖЕРА З ПОКРАЩЕНИМ РІВНЕМ ЗАХИСТУ 23
2.1 Розробка покращеної архітектури та безпеки додатку 23
2.2 Розробка алгоритму роботи додатку 26
2.3 Реалізація функціоналу стеганографії та штучного інтелекту 30
2.4 Висновки 32
3 ПРОГРАМНА РЕАЛІЗАЦІЯ МЕСЕНДЖЕРА 34
3.1 Розробка клієнтської частини 34
3.2 Розробка серверної частини 41
3.3 Розгортання інфраструктури для хостингу додатку 45
3.4 Тестування застосунку 47
3.5 Висновки 49
ВИСНОВКИ 50
ПЕРЕЛІК ПОСИЛАНЬ 51
ДОДАТКИ 53
Додаток А. Технічне завдання 54
Додаток Б. Лістинг 58
Додаток В. Ілюстраційні матеріали 70

ВСТУП

Актуальність даної розробки полягає в покращені безпеки під час обміну конфіденційними даними, що зумовлено покращеним алгоритмом встановлення з'єднання, використанню додаткового тунелю з використанням наскрізного шифрування для з'єднання між клієнтом і сервером а також тунелю для передачі даних від одного користувача до іншого. Також присутні стеганографічні методи захисту інформації як додатковий степінь захисту в разу компрометації з'єднання між клієнтом і сервером та обома користувачами.

Для багатьох користувачів які займаються журналістикою, громадською активністю та іншим, необхідно мати надійний спосіб спілкування який забезпечить достатній рівень захисту.

Актуальність теми посилюється в силу постійного розвитку цифрових технологій та збільшенню потужності електронно обчислювальної техніки, яка в свою чергу може використовуватись для реалізації атак для цифрові канали зв'язку.

Метою роботи є розробка та програмна реалізація месенджера для конфеденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, технологій стеганографії та моделей штучного інтелекту для маскування прихованих даних.

Поставлено та вирішено наступні задачі:

– аналіз існуючих рішень для безпечної комунікації з використанням наскрізного шифрування, визначення їх переваг і недоліків;

– аналіз алгоритмів для реалізації наскрізного шифрування, для передачі повідомлень;

– проектування архітектури та інфраструктури додатку для безпечного розгортання та роботи;

– реалізація абстракції над модулями шифрування, які будуть використовуватись для полегшеної взаємодії з основним модулем;

– реалізація мережевого з’єднання на основі SSL для побудови безпечного каналу зв’язку;

– тестування безпеки та функціоналу розробленого додатку.

Об'єктом дослідження є процес обміну конфіденційною інформацією між співрозмовниками.

Предметом дослідження є розроблені алгоритми шифрування та стеганографії, наскрізне шифрування та стандарти протоколів обміну даними для реалізації безпечного обміну інформацією.

Практичне значення роботи полягає у можливості безпечного обміну конфіденційною інформацією між співрозмовниками в не залежності від безпеки середовища в якому відбувається комунікація. Це дозволить зберігати конфіденційність даних та підвищити рівень безпеки під час обміну інформацією.

1 ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ТЕХНОЛОГІЙ

1.1 Аналіз алгоритмів AES та RSA

AES – сучасник алгоритм шифрування який було опубліковано національним інститутом стандартів і технологій в 2001 році та в 2016 році набув чинності ДСТУ ISO/IEC 18033-3:2015 [1].

AES представляє собою симетричний алгоритм шифрування який може мати розміри блок 128, 192 або 256 бітів. Він є одним з варіантів шифру Rijndael та став на заміну алгоритму DES який був створений в 1977році [2].

На сьогодні AES є одним з найкращих симетричних алгоритмів які дозволяють швидко шифрувати та дешифрувати дані та забезпечувати надійний рівень захисту. Даний алгоритм та його різновиди використовуються в великій кількості додатків для забезпечення безпеки даних, збереження конфіденційності та тривалого зберігання.

AES може мати різні розміри блоків, завдяки чому відрізняється й кількість раундів:

– AES-128 використовує 128-бітний ключ шифрування, що забезпечує 10 раундів шифрування. Маючи понад 3,4 квадрильйона потенційних комбінацій клавіш, AES-128 забезпечує достатній захист для багатьох повсякденних програм, таких як мережі Wi-Fi і служби обміну повідомленнями;

– AES-192 використовує довший 192-бітний ключ, збільшуючи кількість раундів шифрування до 12. Це пропонує приблизно 6,2 секстильйона можливих ключів, що робить його ідеальним для конфіденційних організаційних мереж і передачі файлів;

– AES-256 використовує найнадійніший 256-бітний ключ із 14 раундами шифрування та приблизно 1,1 септильйона потенційних ключів. Завдяки своєму величезному ключовому простору AES-256 зазвичай реалізується для секретних державних комунікацій і шифрування систем критичної інфраструктури.

Також є різні реалізації алгоритму AES, що дозволяє обрати найкращу реалізацію під конкретне завдання:

– ECB (Electronic Codebook) розділяє відкритий текст на блоки, кожен блок шифрується незалежно за допомогою алгоритму AES. Цей режим простий і розпаралелюваний, але він непридатний для шифрування великих обсягів даних або коли дані повторюються, оскільки це може призвести до шаблонів у зашифрованому тексті;

– CBC (Cipher Block Chaining) XOR кожен блок відкритого тексту з попереднім блоком зашифрованого тексту перед шифруванням. Це вводить залежність між блоками, що робить його більш безпечним, ніж ECB, проти атак аналізу шаблонів. Вектор ініціалізації (IV) використовується для шифрування першого блоку, а зашифрований текст кожного блоку використовується як IV для наступного блоку;

– CTR (Counter) перетворює блоковий шифр AES у потоковий шифр. Він використовує значення лічильника в поєднанні з унікальним nonce (числом, що використовується один раз) як вхідні дані для алгоритму AES. Отриманий результат обробляється XOR разом з відкритим текстом для отримання зашифрованого тексту. Режим CTR дозволяє паралельне шифрування та дешифрування та зазвичай використовується в сценаріях, коли потрібен довільний доступ до зашифрованих даних;

– OFB (Output Feedback) також перетворює AES на потоковий шифр. Він генерує потік ключів шляхом шифрування IV за допомогою AES, а потім XOR потоку ключів із відкритим текстом для отримання зашифрованого тексту. На відміну від CTR, OFB вимагає послідовного шифрування та дешифрування, оскільки помилки при передачі можуть поширюватися на наступні блоки.

– CFB (Cipher Feedback) дуже схожий на OFB, але працює з меншими одиницями, зазвичай окремими байтами. Він шифрує результат IV та XOR за допомогою відкритого тексту для створення зашифрованого тексту. Потім потік ключів генерується шляхом шифрування попереднього блоку зашифрованого тексту. Як і OFB, CFB вимагає послідовної обробки;

– GCM (Galois/Counter Mode) поєднує шифрування AES із криптографічною хеш-функцією GHASH. Це забезпечує як конфіденційність, так і цілісність даних. Режим GCM зазвичай використовується в мережевих протоколах зв’язку та підходить для паралельної обробки.

На рисунку 1.1.1 можна побачити приклад роботи AES алгоритму, обов’язковою вимогою якого є збереження ключа в таємниці, оскільки його компрометація несе за собою можливу компрометацію усього каналу зв’язку.

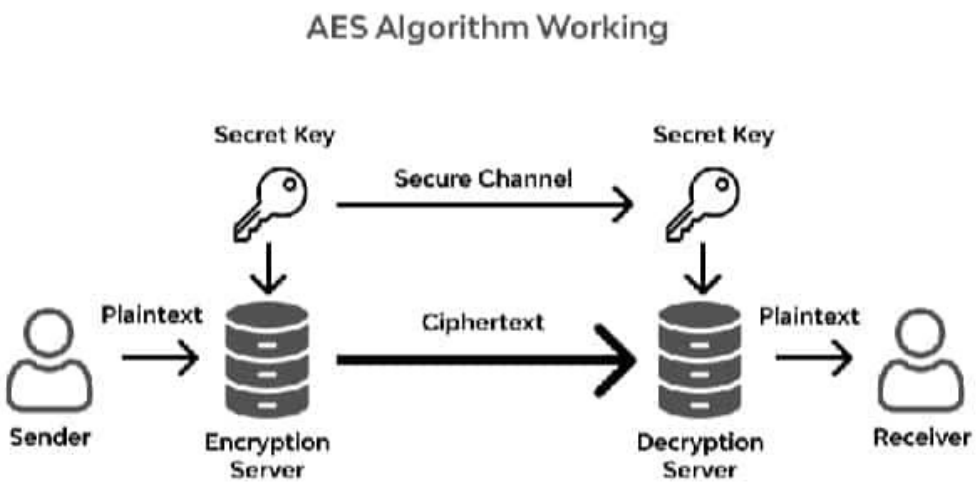

Рисунок 1.1.1 – схема роботи AES алгоритму [3]

З переліку найпоширеніших типів реалізації можна зробити висновок, що AES алгоритм можна застосувати в великій кількості різноманітних технічних завданнях [4].

RSA – асиметричний алгоритм шифрування, основною відмінністю якого є використання в роботі двох видів ключів. Дана особливість дозволяє використовувати даний алгоритм в ненадійних каналах зв’язку. Даний алгоритм було описано в 1977 році. RSA можна використовувати в різних сферах, від передачі даних по незахищеній мережі до цифрового підпису. RSA має великий попит завдяки використанню в роботі публічного та приватного ключа, що дозволяє передати співрозмовнику відкритий ключ, який потім він буде використовувати для шифрування повідомлень які надсилає власнику ключа та дане повідомлення неможна прочитати не маючи приватного ключа.

Робота алгоритму має кілька загальних етапів які відрізняються від симетричних алгоритмів:

– генерація ключа, ціллю цього етапу є отримання експоненти приватного ключа (оберненого за модулем числа) яке визначається за допомогою розширеного алгоритму Евкліда;

– розподіл ключа, під час цього етапу отримуються приватний та публічний ключ на основі попередньо отриманих даних під час виконання етапу генерації ключа;

– шифрування, під час даного етапу використовується відкритий ключ який шифруються повідомлення;

– дешифрування, під час даного етапу використовується приватний ключ аби отримати інформацію яка була зашифрована.

Схема використання RSA алгоритму зображена на рисунку 1.1.2, на якому можна побачити, що немає необхідності передачі публічного ключа по захищеному каналу зв’язку, проте приватний ключ має зберігатись лише в його власника.

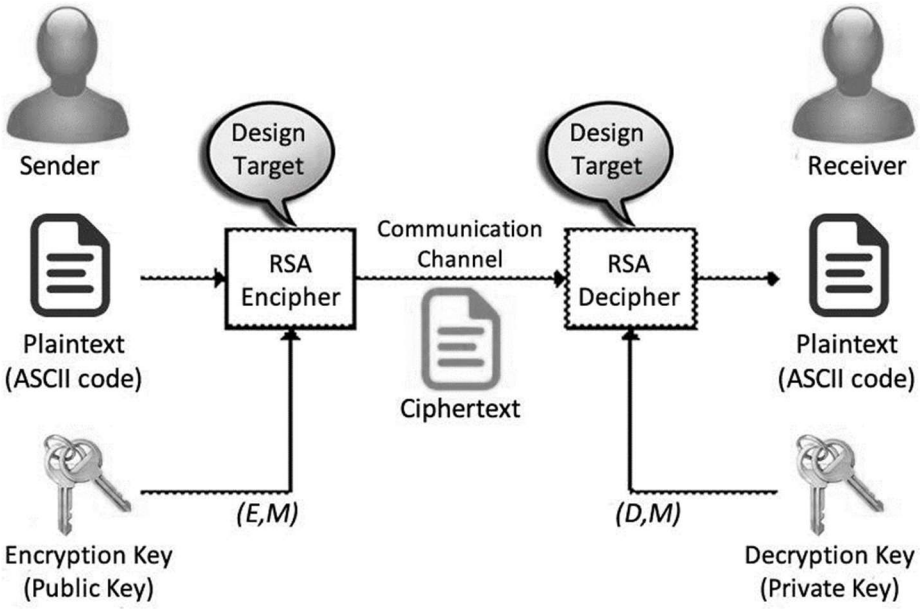

Рисунок 1.1.2 – схема використання RSA алгоритму [5]

Використовувати RSA можна з різною довжиною ключа. На сьогодні безпечною довжиною вважається 4096 біт. До недоліків алгоритму RSA можна

віднести його швидкість роботи яка є в рази довшою в порівнянні з симетричними алгоритмами шифрування та залежність максимальної довжини повідомлення до довжини ключа, тобто довжина повідомлення не може перевищувати довжину ключа [6].

1.2 Концепція наскрізного шифрування

Наскрізне шифрування є важливою частиною сучасних мереж, оскільки дозволяє безпечно встановлювати зв’язок між клієнтами та передавати необхідну інформацію. В наскрізному шифруванні використовується два алгоритми, перший це асиметричний алгоритм який в своїй роботі використовує публічний та приватний ключ та використовується для встановлення безпечного зв’язку, другим алгоритмом є симетричний та використовується він для передачі повідомлень після використання асиметричного для встановлення з’єднання між клієнтами. Сьогодні наскрізне шифрування використовується або є E2EE використовується в великій кількості сфер, серед яких є:

– Охорона здоров’я: E2EE часто використовується в телемедицині та системах електронних медичних записів для захисту даних пацієнтів від несанкціонованого доступу.

– Фінанси банки та фінансові установи використовують E2EE для захисту онлайн-транзакцій і спілкування між клієнтами та співробітниками;

– Комунікація програми обміну повідомленнями, служби електронної пошти та інструменти для відеоконференцій використовують E2EE для забезпечення безпечного спілкування між користувачами;

– Освіта платформи онлайн-навчання та системи керування даними студентів використовують E2EE для захисту даних студентів і викладачів;

– Електронна комерція E2EE використовується для захисту платіжної інформації та даних клієнтів під час онлайн-транзакцій.

Завдяки своїй простоті реалізації та багатозадачності наскрізне шифрування (рис 1.2.1) є популярним підходом до побудови конфіденційної мережі [7].

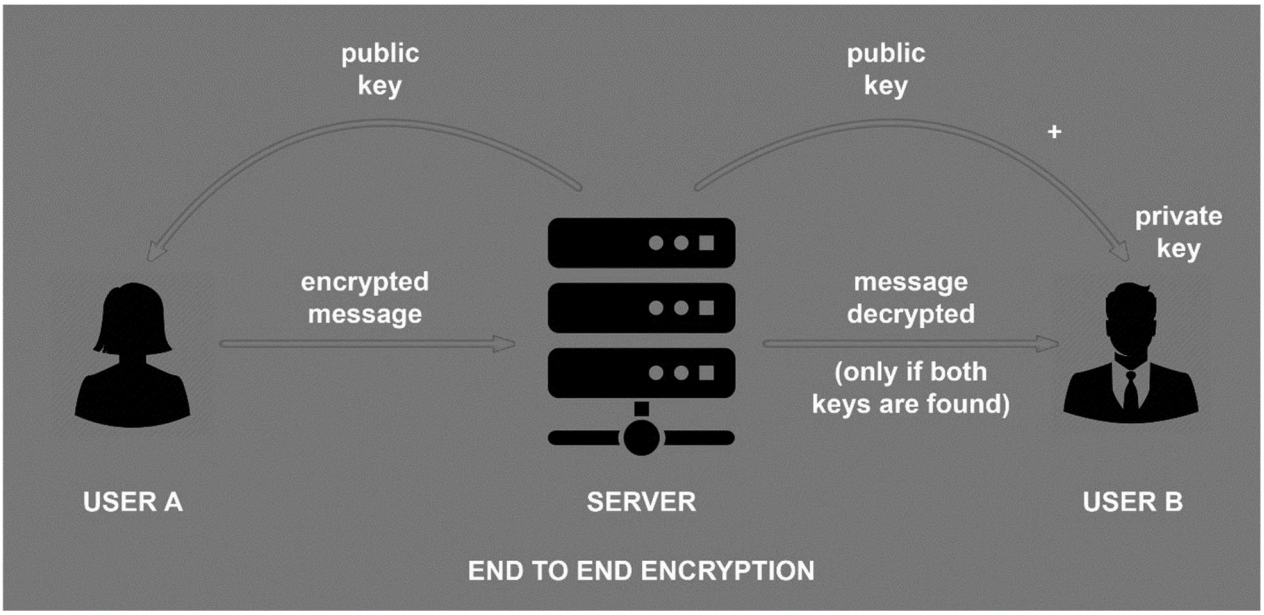

Рисунок 1.2.1 – схема роботи наскрізного шифрування [8]

Також існують кращі практики щодо побудови мережі де використовується наскрізне шифрування:

– для реалізації використовувати перевірені та надійні алгоритми шифрування;

– генерувати та використовувати унікальні ключі для кожного користувача;

– мати надійних сховок ключів які зберігаються на серверній та клієнтській частині додатку;

– реалізація perfect forward secrecy (PFS), мета якої забезпечити безпеку даних які зашифровані іншими ключами при компрометації одного, для кожного сеансу зв’язку використовуються нові ключі;

– використання методів верифікації користувачів для запобігання несанкціонованого доступу;

– користувач повинен мати змогу впевнитись в роботі шифрування, що воно ввімкнено і працює правильно.

Оскільки даний підхід є критичною складовою безпеки каналу зв’язку, слід ретельно вивчати алгоритми за допомогою яких виконується реалізація даного підходу. Також важливим є те, як зберігаються приватні ключі, оскільки

компрометація будь якого ключа може призвести до компрометації каналу або каналів зв’язку [7].

Розглянемо аналоги які є на ринку

Таблиця 1 – Порівняння аналогів

Назва	Наскрізне шифрування	Анонімність користувача	Використання стеганографії
Telegram	-	-	-
Viber	+	-	-
Discord	+	+	-
Signal	+	+	-
WhatsApp	-	-	-
Розробляємий додаток	+	+	+

Проаналізувавши дані, наведені в таблиці 1, можна побачити, що аналоги не мають можливості використанням стеганографічних методів в самому додатку. Також в більшості анонімність користувача, оскільки є необхідні підтвердження електронної пошти або телефону. Також в деяких відсутнє наскрізне шифрування, що становить загрозу можливої компрометації повідомлень.

1.3 Приховування секрету в строкових повідомленнях

Стеганографія є одним із методів, який використовується для приховування конфіденційної інформації від неавторизованих сторін. Цей метод захищає конфіденційну інформацію, таку як зображення чи текст, приховуючи її у звичайних на вигляд файлах або повідомленнях, що ускладнює доступ до неї неавторизованих користувачів. Стеганографія, як метод захисту інформації, використовується з 440 року до нашої ери. Однак використання цього методу в сучасній кібербезпеці почалося в кінці 20 століття [9].

Існує безліч видів стеганографії завдяки яким можна приховувати секретне повідомлення (секрет) в різних типах даних:

- приховування інформації в текст;
- приховування інформації в зображення;
- приховування інформації в відео;
- приховування інформації в аудіо;
- приховування інформації в сигнали мережі.

Приховування секретного повідомлення в текст відбувається завдяки використанню прихованих символів в таблиці Unicode. Для реалізації додатку буде використано алгоритм StegCloak який виконує низку стиснень для мінімізації тексту який треба буде приховати, шифрує дані за допомогою AES алгоритму та після за допомогою прихованих символів вбудовує зашифроване повідомлення в відкритий текст.

StegCloak може захистити повідомлення за умови збереження в приватного ключа в секреті, чим реалізує принцип Принцип Керкгоффса [10].

За для кращою безпеки за умови використання коротких та слабких паролів, додається випадковий набір бітів, ціль яких запобігти атакам в аналізом кількох зашифрованих текстів, також для цього використовується AES алгоритм в реалізації CTR, ця реалізація дозволяє зменшити кількість бітів для приховування повідомлення в тексті. Загальну схему роботи алгоритму StegCloak зображено на рисунку 1.3.1 та рисунку 1.3.2.

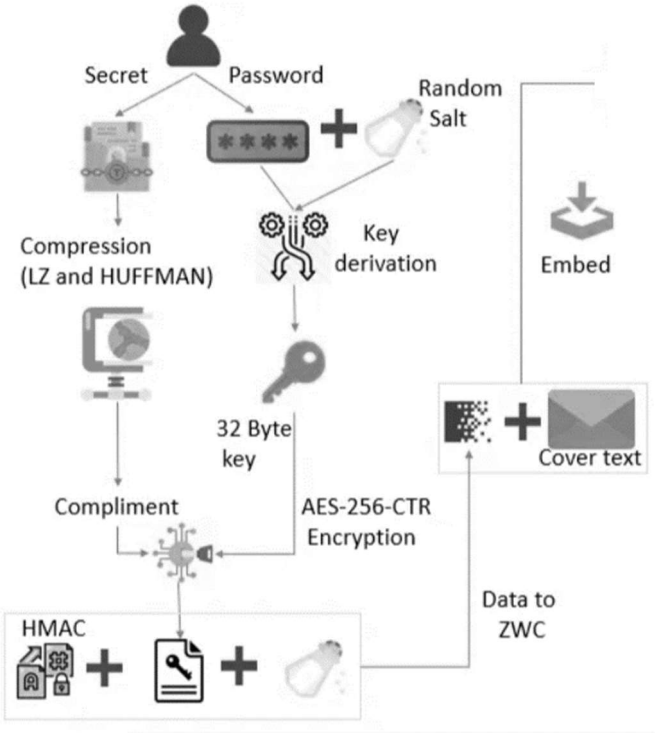

Рисунок 1.3.1 – схема роботи алгоритму StegCloak при приховуванні даних

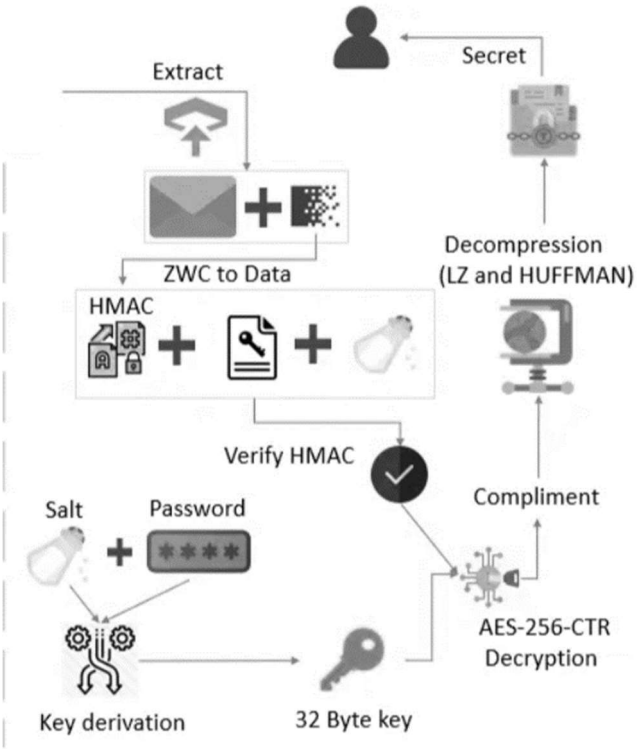

Рисунок 1.3.2 – схема роботи алгоритму StegCloak при розкриття даних

Завдяки даному алгоритму можна передавати захищені повідомлення в вигляді тексту який не несе важливої інформації, та може бути використаний в чаті з наскрізним шифруванням як додатковий вид захисту за умов компрометації

каналу зв’язку. Повідомлення створені за допомогою даного алгоритму коректо відображаються та не є помітними в різних месенджерах, чатах та інших системах які дозволяють поширювати текстову інформацію [11].

1.4 Штучний інтелект та його застосування для маскування даних

Штучний інтелект це система яка виконує завдання яке традиційні для людини. Системи проходять навчання на великому обсягу даних, завдяки чому модель формує певні шаблони і патерни які в майбутньому використовує для виконання поставлених задач. Також система може навчатись на даних які формує саме в процесі виконання завдання за для покращення своєї роботи.

Штучний інтелекк працює на основі алгоритмів та даних які йому були надані, за допомогою певних алгоритмів визначаються закономірності в даних, що в свою чергу після навчання, дозволяє моделі штучного інтелекту розпізнавати зображення, генерувати текст та виконувати низку інших задач [12].

Основним підходом для побудови системи штучного інтелекту є машинне навчання, коли за допомогою певних алгоритмів виконується аналіз великої кількості даних. Для виконання аналізу використовуються статичні методи які дозволяють поступово навчитись моделі виконувати поставлену задачу. Існує два види навчання:

– контрольоване, де є очікувані вихідні дані, що досягається шляхом позначеного набору даних;

– неконтрольоване, де вихідні дані є невідомими оскільки використовуються непозначений набір даних при запуску алгоритму.

Схематичне зображення роботи штучного інтелекту на рисунку 1.4.1.

Машинне навчання реалізається за допомогою нейронних мереж, що має в собі ряд алгоритмів які імітують роботу людського мозку, дані алгоритми пов’язані між собою та обробляють інформацію разом, передаючи її один одному [13].

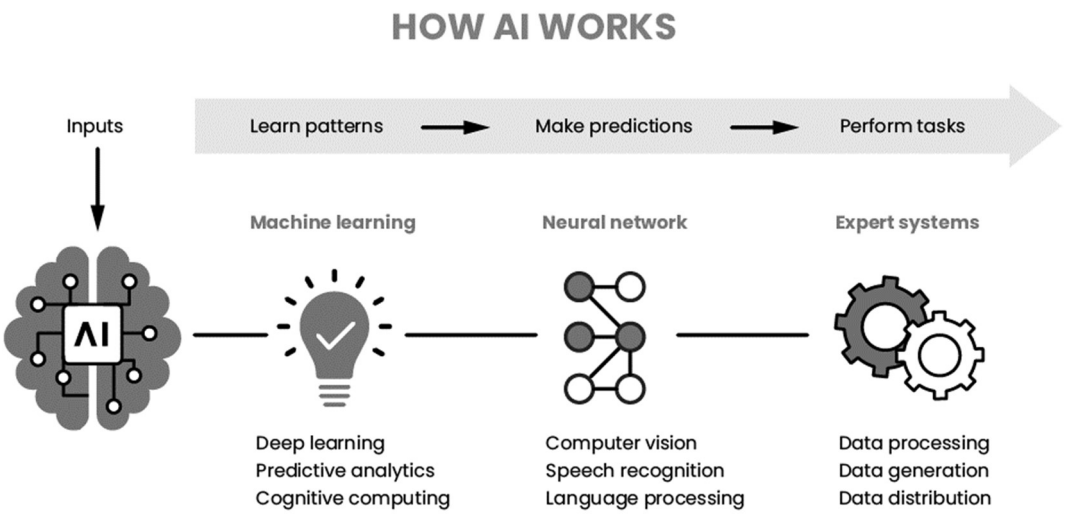

Рисунок 1.4.1 – схематичне зображення роботи штучного інтелекту [14]

Штучний інтелект отримав свою популярність в великій кількості галузей завдяки можливості швидко виконувати складні операції над якими в людини йде значно більше часу. До таких сфер діяльності належить:

– медицина, за допомогою ШІ аналізують МРТ та інші знімки для аналізу пухлин та інших хвороб, виявлення яких є тривалим та складним процесом. Також штучний інтелект використовується в якості асистента лікаря для полегшення роботи та мінімізації помилок при огляді пацієнтів або під час лікування;

– охоронні системи, з використанням штучного інтелекту є можливість аналізувати поведінку людини та виявляти потенційних порушників закону, також ШІ використовується для пошуку людини за обличчям, поведінкою та іншими аспектами які є унікальними для кожної людини;

– перевірка систем на вразливості, нещодавно AI почали використовувати для пошуку вразливостей в комп’ютерних мережах для покращення безпеки організації в цифровому просторі;

– написання тексту, ШІ використовується для написання текстів відповідно до запиту який було зроблено людиною, це допомагає автоматизувати низку робіт пов’язаних з описам товару, створені постів в мережах та багато іншого;

– створення зображень, як допомога графічним дизайнерам, Штучний інтелект використовується для створення прикладів зображень які потім використовуються для створення нового витвору мистецтва.

– автоматизація бізнес процесів, за допомогою AI можна підвищити продуктивність слабких або тяжких ланок на виробництві, також за допомогою ШІ можна зменшення тривалості виготовлення певної продукції а також підняти її якість;

– фінансовий моніторинг, використовується для пошуку і аналізу фінансових махінацій та завчасного реагування на інциденти, що дозволяє підвищити безпеку банківської системи.

Ці сфери вимагають великих витрат по часу на обробку великої кількості інформації, яку не в силах зробити людина але з чим добре справляються системи штучного інтелекту, мінімізуючи можливість виникнення помилки при обробці даних для отримання результату.

Для реалізації програмного застосунку, буде використовуватись мовна модель ChatGPT оскільки за допомогою текстового запиту, дана мовна модель генерує необхідне текстове повідомлення.

ChatGPT розроблений компанією OpenAI та представлений перший зразок в 2022 році. В своїй роботі ChatGPT-2 використовує 1.5 міліарда параметрів, що прирівнюється до 6 гігабайт даних. Від кількості параметрів напряму залежить якість згенерованого тексту який [15].

На рисунку 1.4.1 зображено схемо роботи ChatGPT від процесу самого навчання моделі штучного інтелекту до початку процесу її використання.

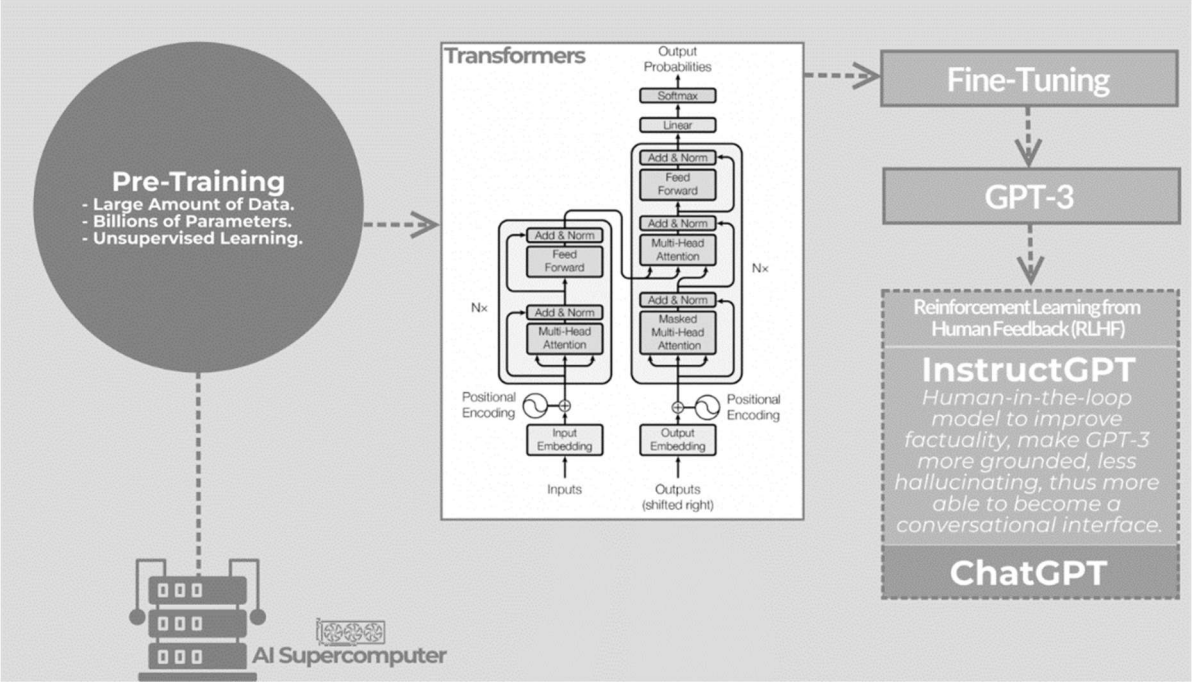

Рисунок 1.4.1 – схема роботи ChatGPT

Штучний інтелект є потужним інструментом за допомогою якого можна оптимізувати велику кількість складних і важливих галузей, AI використовується як помічник людині для того аби спростити і пришвидшити виконання певного завдання. Також слід враховувати, що ШІ може бути вкрай небезпечним за умови відсутності належного контролю під час його використання звичайними користувачами, оскільки дана технологія імітує роботу людського мозку з можливістю саморозвитку.

1.5 Висновки

В даному розділі виконано аналіз предметної області, проаналізовано та ознайомлено з можливостями приховування даних в текстовому повідомленні а також виконаний аналіз мовної моделі для автоматичної генерації тексту який буде використовуватись як контейнер для повідомлення з прихованим вмістом.

Було розглянуто протоколи обміну даними та схема для побудови безпечного каналу зв’язку для майбутнього безпечного обміну повідомленнями.

Для розробки та практичної реалізації необхідно встановити відповідні задачі, а саме:

– реалізації наскрізного шифрування, для передачі повідомлень;

– проектування архітектури та інфраструктури додатку для безпечного розгортання та роботи;

– реалізація абстракції над модулями шифрування, які будуть використовуватись для полегшеної взаємодії з основним модулем;

– реалізація мережевого з’єднання на основі SSL для побудови безпечного каналу зв’язку;

– тестування безпеки та функціоналу розробленого додатку.

Дані задачі є важливими для виконання, оскільки містять в собі важливі складові додатку, які прямо впливають на його безпеку.

Наступні розділи будуть присвячені проектуванню та практичній реалізації месенджера на основі отриманих теоретичних знань. Буде розроблено архітектуру системи, протокол обміну повідомленнями, а також програмну реалізацію клієнтської та серверної частин. Особлива увага буде приділена інтеграції розглянутих у першому розділі технологій.

2 ПРОЕКТУВАННЯ МЕСЕНДЖЕРА З ПОКРАЩЕНИМ РІВНЕМ ЗАХИСТУ

2.1 Розробка покращеної архітектури та безпеки додатку

Для покращення рівня безпеки додатку, буде виконано програмну реалізацію клієнт серверного з'єднання з виконанням наскрізне шифрування усіх даних та мати можливість для приховування текстового повідомлення в стеганографічному контейнері, який також є текстом, що дозволить підвищити безпеку і приватність спілкування та є додатковим захистом в разі компрометації каналу зв'язку або загрозам "Shoulder Surfing" підглядання через плече.

При розробці додатку, в його архітектуру закладено сучасні технології які активно підтримуються провідними компаніями. Дані технології активно тестуються та виплавляються вразливості різних ступенів.

Буде використано мову програмування TypeScript, дана мова набрала велику популярність завдяки зручності а також використання її при створенні клієнтської та серверної частини суттєво пришвидшить розробку. TypeScript надає можливість задавати типізацію змінних, класів та функцій а також оголошувати інтерфейси, завдяки чому досягається чистота коду та ускладнює запис даних невказаного типу в змінну або іншу конструкцію мови програмування. TypeScript є інтерпритованою мовою програмування, завдяки чому запуск додатку не залежить від платформи.

Також при реалізації удосконаленого месенджеру буде використано такі бібліотеки як Next.js для створення клієнтської частини додатку на якій буде виконуватись основна логіка додатку, пов'язана з криптографічними та стенографічними обчисленнями та Node.js для реалізації серверу який буде виконувати операції з базами даних та з штучним інтелектом.

Буде використано реляційну базу даних PostgreSQL для постійного зберігання даних.

Також буде виконана інтеграція з моделлю штучного інтелекту від OpenAI а саме ChatGPT для генерації тексту який буде використовуватись для приховування повідомлень.

Для розгортання та використання месенджеру буде розгорнуто інфраструктуру з використанням інструменту з відкритим вихідним кодом Jenkins, для контейнеризації та ізольованого запуску Docker. Інфраструктура додатку буде розгорнута на серверах AWS з допомогою Terraform для стабільної роботи та безпеки додатку. Terraform це потужний інструмент з відкритим вихідним кодом, який дозволяє налаштовувати інфраструктура хмарних провайдерів за допомогою коду який попередньо був написаний. Даний підхід дозволяє швидко вносити необхідні зміни в інфраструктура платформи та полегшує масштабування додатку.

Для хостингу додатку буде розгорнуто інфраструктуру в AWS яка в собі буде містити Virtual private cloud (VPC) з двома підмережами, серед яких буде приватна та публічна. Приватна мережа не зможе мати виходу в інтернет за для збереження безпеки бази даних та отримати до сервісів які будуть знаходитись в публічній мережі можна тільки через basios сервер, публічка мережа в свою чергу буде мати повний доступ в інтернет. В публічній мережу будуть знаходитись клієнтська та серверна частини додатку. Дану схему зображено на рисунку 2.1.1.

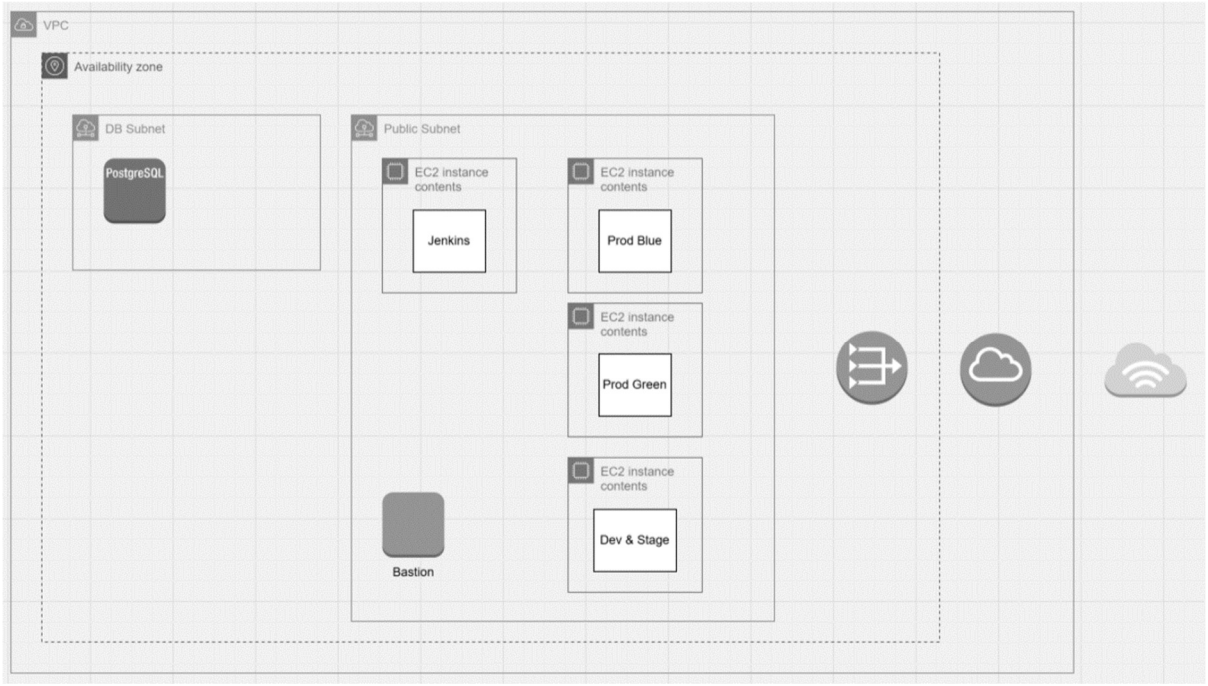

Рисунок 2.1.1 – схема інфрастуктури додатку

Для виходу в інтернет буде використано NAT Gateway. Та для стабільної роботи буде використано підхід Blue Green deploy перевагою якого є можливість використовувати додаток під чат оновлення або при виникненні критичних помилок.

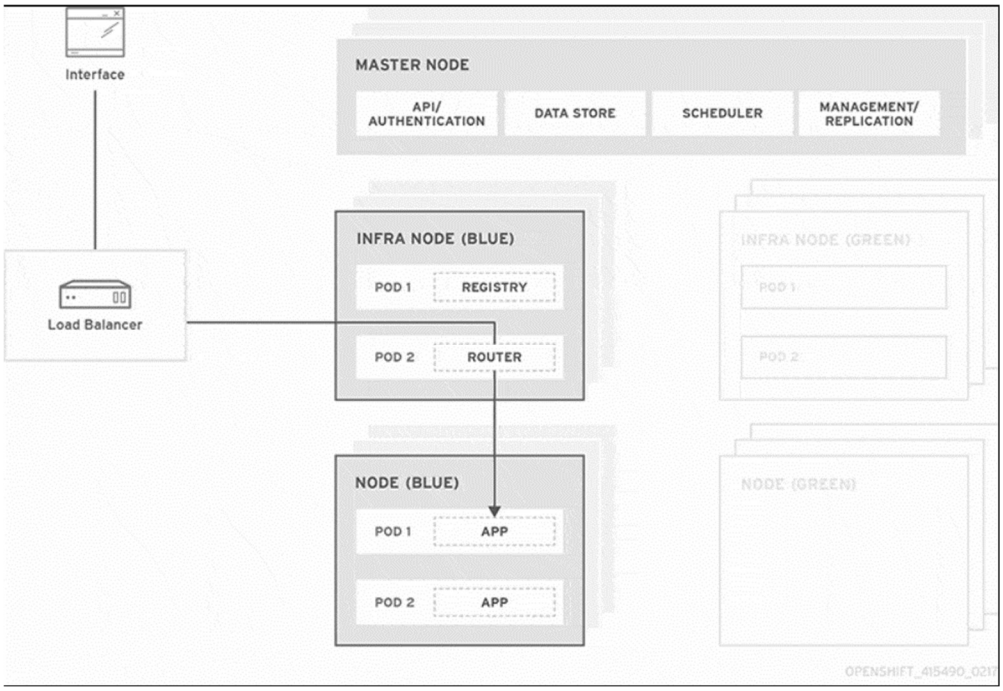

Рисунок 2.1.2 – схема роботи Blue Green deploy [16]

Даний вид розгортання дозволяє розгорнути додаток на сервері який в майбутньому будуть використовувати користувачі та повноцінно виконати тестування усього необхідного функціоналу. Після проведення тестів в налаштуваннях перенаправляють трафік додатку з застарілої версії на нову, тим самим основний трафік користувачів прямує до оновленого серверу. З другим сервером можливе використання двох шляхів. Перший це залишити стару версію аби в разі виникнення помилки в оновленій версії або ж виконати оновлення для другого серверу також [17]. Для реалізації даного типу доставки додатку буде виконано розгортання двох незалежних VPC разом з базами даних, серверами та підмережами. Перемикання буде відбуватись за допомогою налаштування cloudflare (рис 2.1.3), де тип запису A направляє до ip адреси, таким чином при необхідності перенаправити трафік користувачів, відбувається зміна ip адреси в записі типу A.

Type	Name	Content	Proxy status	TTL	Actions
A	[redacted]-site	[redacted]	Proxied	Auto	Edit
CNAME	[redacted]-site-dev	[redacted]	Proxied	Auto	Edit
CNAME	[redacted]-site-stage	[redacted]	Proxied	Auto	Edit

Рисунок 2.1.3 – налаштування cloudflare

Такий підхід до побудови архітектури додатку дозволить швидко та безпечно вводити нові оновлення, оскільки перед перемиканням трафіку користувачів є можливість перевірки працездатності всіх серверів та впевнитись в їх коректній роботі.

2.2 Розробка алгоритму роботи додатку

Обмін повідомленнями буде виконуватись за допомогою протоколу обміну даних WebSocket. Даний протокол дозволяє в реальному часі обмінюватись даними між клієнтом і сервером, що в свою чергу дозволить серверу повідомляти користувача коли йому прийшло нове повідомлення (рис 2.2.1).

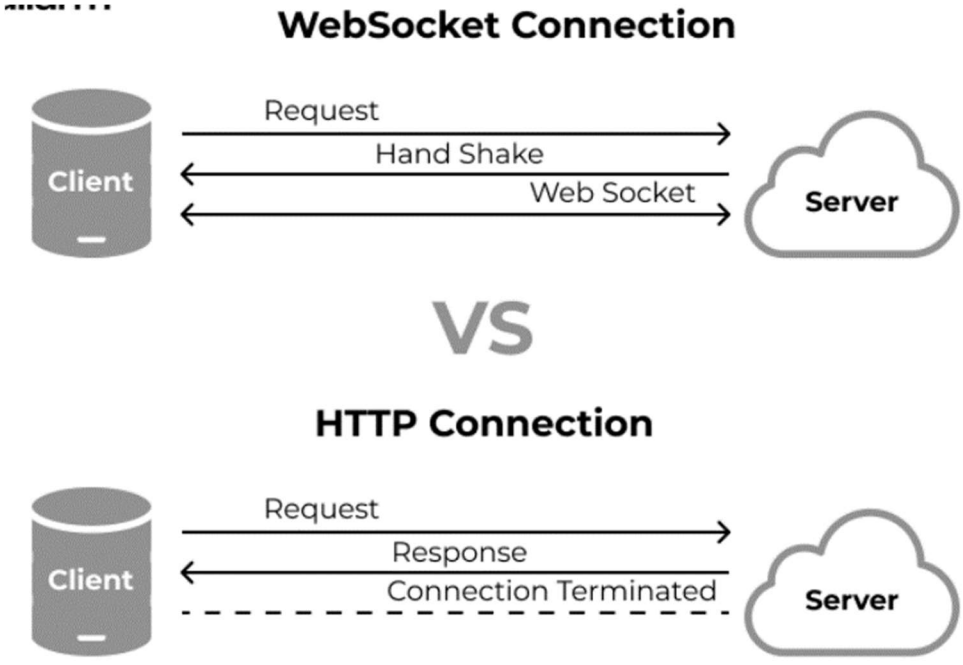

Рисунок 2.2.1 – приклад роботи WebSocket протоколу в порівнянні з HTTP [18]

Для реалізації наскрізного шифрування та покращати захист даних для авторизації користувача, буде встановлюватись запит на HTTP маршрут з метою отримання відкритого RSA ключа серверу. Після отримання ключі, клієнт генерує

ключ для AES алгоритму, який буде використовуватись для шифрування усього трафіку користувача. Після генерації ключа та виконання його шифрування за допомогою відкритого RSA ключа, встановлюється з’єднання з сервером за допомогою WebSocket де першим повідомленням передається зашифрований ключ.

Вигляд тіла повідомлення:

```
{
  status: 200,
  route: session.key.set,
  data: {
    aesKey: “AES Key”
  }
}
```

Наступним кроком є підтвердження сервером ключа та в разі успішної перевірки користувач може надіслати свої дані для авторизації та увійти в свій акаунт (рис 2.2.2).

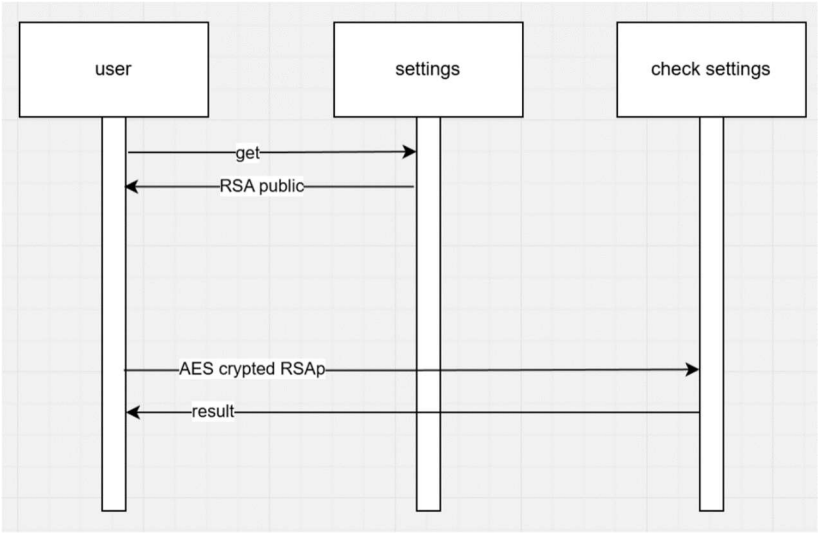

Рисунок 2.2.2 – життєвий цикл обміну ключами

Також для спрощення розробки та відлагодження додатку буде можливість вимкнути модуль, який забезпечує обмін ключами для встановлення з’єднання з наскрізним шифруванням (рис 2.2.3).

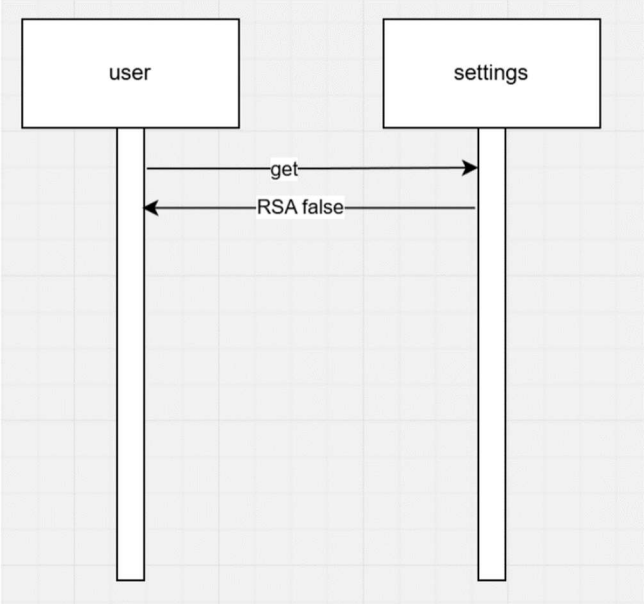

Рисунок 2.2.3 – життєвий цикл авторизації за умови вимкненого модуля обміну ключами

Після узгодження кроків для авторизації користувача в системі необхідно узгодити загальну структуру повідомлень та помилок, які будуть відправляти як клієнт так і сервер один одному.

Перш за все необхідно поле, яке вказує до якого маршруту належить певна відповідь або запит, дане поле буде називатись “route” та міститиме строку яка відповідає сутності в базі даних яку змінює чи отримує. Наступне поле це “status”, воно необхідне для повідомлення клієнту про статус виконання запитів, дане поле буде відповідати статусам http методів, де коди з статусами 200 це успішно виконані запити, статуси 300 які відповідають за перенаправлення користувача, 400 некоректний запит зі сторони користувача та 500 які вказують на помилки серверу [19]. Також буде присутнє поле “data”, в якому буде зберігатись тіло відповіді або запиту. При передачі повідомлень з чату, більшість інформації буде знаходитись за ключем data в полі message. В даному полі буде знаходитись текст який передається, час відправки та власник повідомлення, що в свою чергу забезпечить безпеку користувача за умови компрометації бази даних, оскільки не буде відомо які саме та скільки повідомлень було надіслано конкретним користувачем месенджеру.

Після розробки алгоритму для клієнт серверної взаємодії, виконано розробку блок-схеми роботи застосунку (рис 2.2.4).

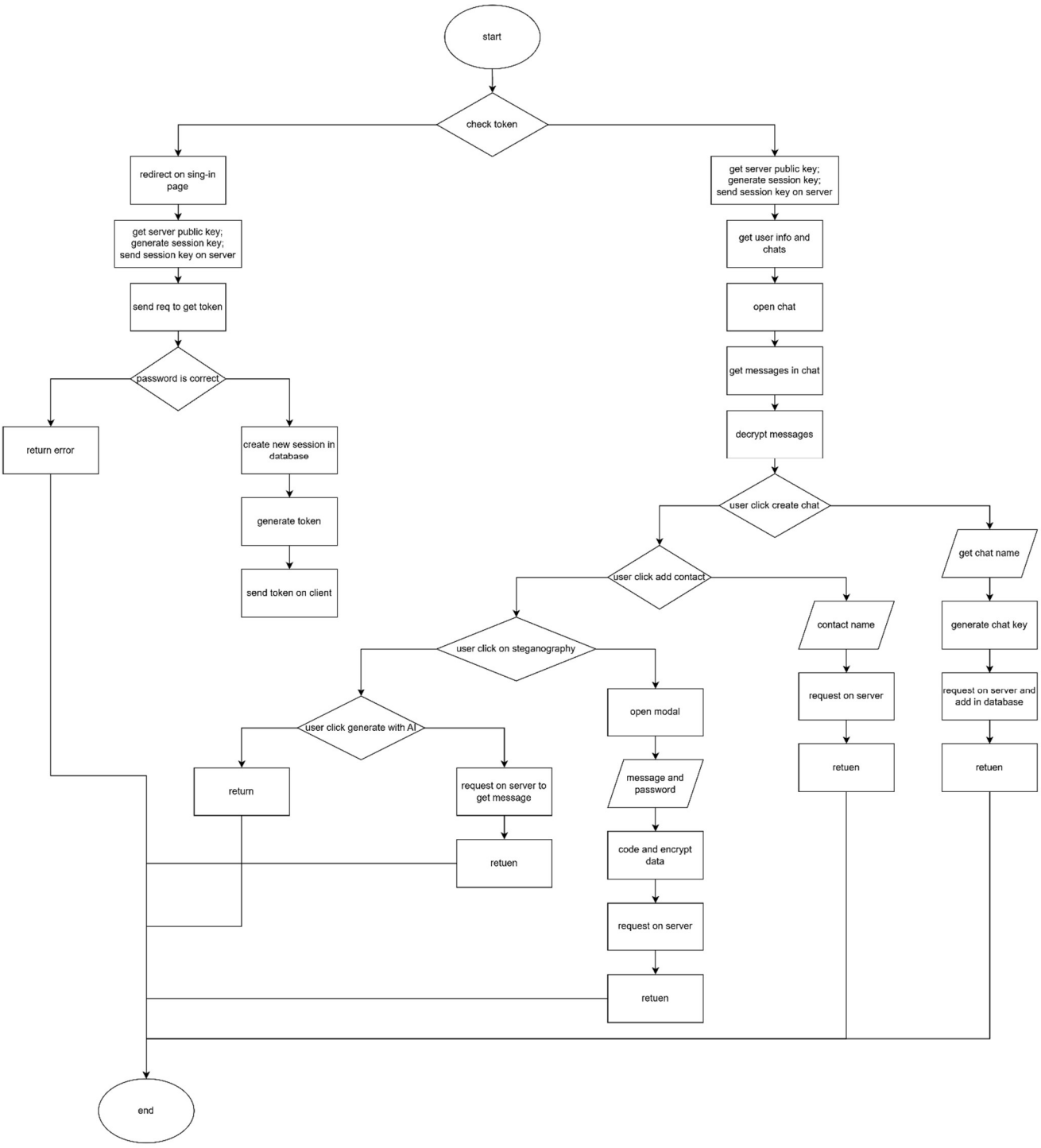

Рисунок 2.2.4 – блок схема роботи застосунку

На рисунку 2.2.4 можна побачити, що робота додатку можлива лише за умови створеного безпечного з’єднання, після якого користувач може

використовувати повний функціонал додатку. Враховані можливості щодо додавання контактів, створення нових чатів, використання стеганографічних методів захисту інформації, які відбуваються тільки на клієнтській стороні та враховано інтеграцію з мовною моделлю.

Першим кроком відбувається перевірка чи з’єднання є безпечним, після чого користувача або перенаправляє на сторінку авторизації або він заходить в свій робочий простір. Після, коли користувач увійшов, він може відкрити чат, додати новий контакт, створити новий чат, відправити повідомлення в відкритому чаті або повідомлення, яке знаходиться в стеганографічному контейнері, також перед відправкою такого повідомлення, він може згенеруватись питання за допомогою штучного інтелекту.

2.3 Реалізація функціоналу стеганографії та штучного інтелекту

Для реалізації генерування повідомлень за допомогою штучного інтелекту, буде використано модель штучного інтелекту від OpenAI а саме chat GPT. Для взаємодії з даною моделлю штучного інтелекту буде використовуватись модуль, який був розроблений компанією OpenAI саме для взаємодії сторонніх сервісів з їх продуктом.

Оскільки ключ який надає доступ до моделі штучного інтелекту має зберігатись на сервері, взаємодія з клієнтським додатком буде виглядати наступним чином:

1) клієнт відправляє запит на сервер додатку через відповідний http запит для генерації тексту;

2) використовуючи модуль від openAI відправляє запит до моделі ШІ з метою генерації тестового повідомлення;

3) сервер за умови отримання результату роботи штучного інтелекту, відправляє цей результат користувачу;

4) користувач отримує згенероване повідомлення яке буде використано для приховування основного тексту повідомлення.

На рисунку 2.3.1 зображено життєвий цикл процесу генерації текстового повідомлення. Також слід враховувати, що штучний інтелект треба обмежувати по довжині відповіді аби не буде надто велике повідомлення.

Також в користувача буде змога самому ввести повідомлення яке потім буде використовуватись для приховування текстової інформації.

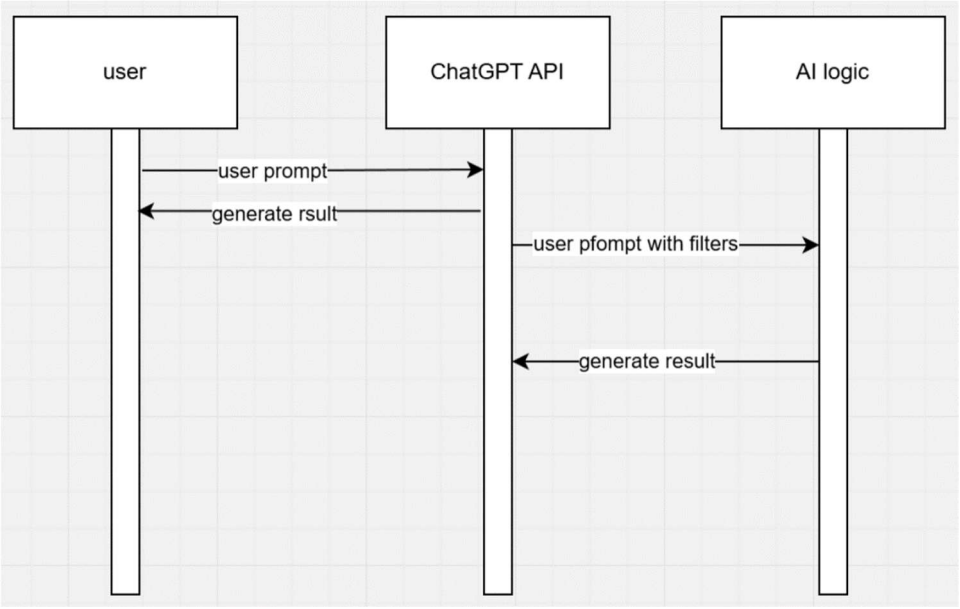

Рисунок 2.3.1 – життєвий цикл генерації тексту по запиту користувача

Після введення або отримання текстового повідомлення яке буде використовуватись як контейнер для зберігання основного повідомлення, необхідно ввести саме основне повідомлення в відповідне поле та за необхідності додати пароль який слугує захистом від випадкового відкриття.

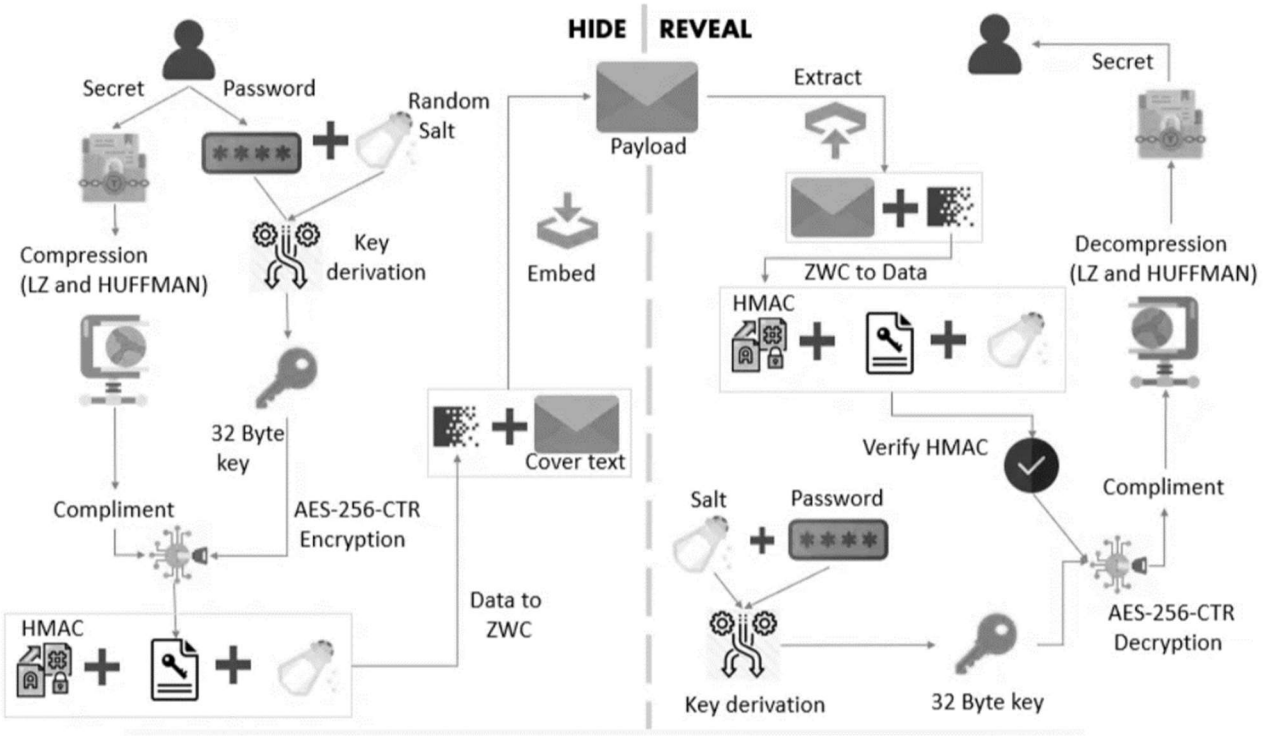

Рисунок 2.3.2 – схема роботи стеганографічного алгоритму StegCloack

Даний алгоритм є криптостійким завдяки використанню AES алгоритму в основі роботи процесу приховування тексту. Також виконується низка стиснень текстових даних які будуть приховуватись в текст, що дозволяє ускладнити криптоаналіз прихованих даних.

2.4 Висновки

В даному розділі було розглянуто важливі аспекти удосконалення месенджера для конфіденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, технологій стеганографії та моделей штучного інтелекту для маскування прихованих даних.

Було розроблено структуру повідомлення, яка буде використовуватись для обміну та зберігання повідомлень між клієнтом і сервером, також було розглянуто роботу алгоритмів шифрування, які використовуються в додатку. Зокрема, досліджено особливості використання комбінації симетричного (AES) та асиметричного (RSA) шифрування для забезпечення як швидкості, так і безпеки

передачі даних. Розглянуто процес генерації та обміну ключами, а також механізми їх безпечного зберігання на клієнтських пристроях.

Було розглянуто взаємодію серверної частини додатку з моделлю штучного інтелекту, основною метою якої є генерація текстових повідомлень, які будуть використовуватись як контейнер для основного повідомлення. Детально проаналізовано архітектуру моделі ШІ, методи її навчання та оптимізації для генерації різноманітних та природних текстів. Також розглянуто механізми фільтрації та перевірки згенерованих текстів для забезпечення їх відповідності вимогам стеганографії.

Для виконання приховування повідомлення в текстових даних, розглянуто алгоритм StagCloack, який дозволяє виконати приховування текстової інформації в середині іншого тексту, використовуючи особливості текстових систем розмітки або ж текстових кодувань.

Також було розроблено процес розгортання додатку на серверах, який буде доступний кінцевим користувачам. Розглянуто переваги, які надає обраний тип доставки додатку, зокрема використання Docker. Додатково розглянуто питання безпеки серверної інфраструктури, включаючи захист від DDoS-атак, вразливостей програмного забезпечення та несанкціонованого доступу.

В наступному розділі буде розглянуто програмну реалізацію додатку за допомогою вище згаданих технологій програмування. Буде детально описано архітектуру додатку, вибір мов програмування та фреймворків, а також процес розробки та тестування окремих компонентів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МЕСЕНДЖЕРА

В даному розділі описано практичну реалізацію клієнтської та серверної частини додатку, побудову структури частин проекту та виконано його тестування.

Під час реалізації додатку використано сучасні технології які суттєво спрощують процес розробки, розгортання та конфігурації додатку.

3.1 Розробка клієнтської частини

Для програмної реалізації буде використано Next.js з мовою програмування TypeScript.

TypeScript є сучасною мовою яка компілюється в JavaScript та дозволяє використовувати строгу типізацію під час розробки. Next.js був обраний завдяки мжливості швидко реалізовувати програмний функціонал а також він має пакет попередньо встановлених бібліотек які полегшують розробку.

Перш за все необхідно ініціалізувати проекти, що робиться командою наведеною в офіційній документації

```
npx create-next-app@latest
```

Після початкової ініціалізації створюється файлова архітектура додатку, яка в свою чергу дозволяє легко орієнтуватись в проекті та закладає потенціал для майбутнього розвитку додатку. Також в Next.js завдяки побудові ієрархічної системи папок можна конфігурувати роутинг в додатку.

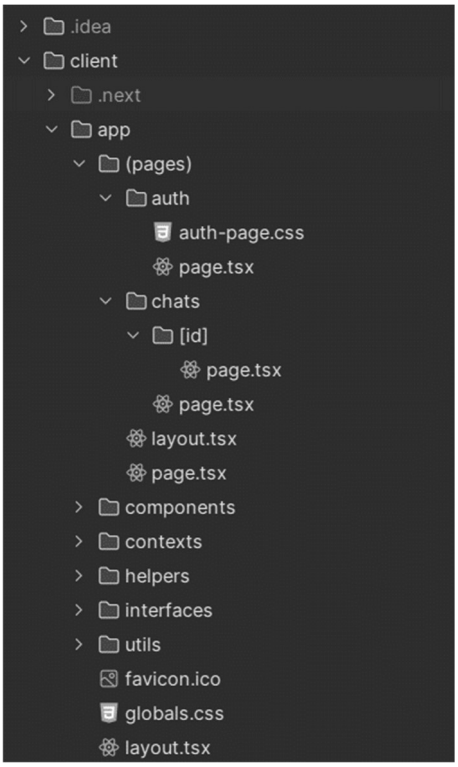

Рисунок 3.1.1 – зображення структури проекту

Після реалізації структури, необхідно переходити до реалізації інтерфейсу користувача.

Реалізації сторінки авторизації виглядає наступним чином:

```
<div className={"auth-page"}>
  <div className={"auth-page__content w-full"}>
    <h1 className={"auth-page__content-title"}>
      {isLogin ? "Sing in" : "Sing up"}
    </h1>
```

Оскільки дана сторінка є універсальною та використовується для реєстрації та авторизації користувача, необхідно виконувати перевірку, що бучить користувач відповідно до змінної isLogin.

```
<div className={"w-full"}>
  {isLogin && (
    <LoginPart
      loginNameValue={loginNameValue}
      loginPasswordValue={loginPasswordValue}
      setLoginNameValue={setLoginNameValue}
      setLoginPasswordValue={setLoginPasswordValue}
    />
  )}
```

```
    {!isLogin && (    <RegPart
       registrationPasswordValue={registrationPasswordValue}
       setRegistrationPasswordValue={setRegistrationPasswordValue}
       registrationNameValue={registrationNameValue}
       setRegistrationNameValue={setRegistrationNameValue}
     />     )}   </div>
  <div className={"w-full"}>
   <button
    className={"w-full btn btn-primary btn-lg btn-block"}
    onClick={handlerSubmit}
   >       submit     </button>   </div>   <div className={"w-full"}>
   <button
    className={"btn btn-link"}
    onClick={() => setIsLogin(!isLogin)}
   >
    {!isLogin ? "go to sing in" : "go to sing up"}
   </button>   </div>  </div></div>
```

Наступним кроком виконується реалізації логіки авторизації та реєстрації в системі:

```
export const AuthProvider: React.FC<IAuthProvider> = ({ children }) => {
  const [isAuth, setIsAuth] = useState(false);
  const [sessionRsa, setSessionRsa] = useState("");
  const [sessionAes, setSessionAes] = useState("");
  const [isRegistered, setIsRegistered] = useState("");
  const [user, setUser] = useState<IUser | undefined>(undefined);

  const { isConnect, addWsListener, removeWsListener, wsSend } =
    useWebSocketContext();
```

Після ініціалізації змінних які будуть зберігати відповідні стани які пов’язані з реєстрацією та авторизацією, необхідно підписатись на події WebSocket для можливості приймати повідомлення від серверу.

```
 useEffect(() => {
  addWsListener("auth.get.rsa.res", handlerGetReaKey);
  addWsListener("auth.set.aes.res", handlerSetSessionAesKey);
  addWsListener("auth.login.res", handlerLogin);
  addWsListener("auth.registration.res", handlerReg);
 }, [, sessionAes]);
    wsSend({ event: "auth.get.rsa" });
 };
 const handlerGetReaKey = (msg: IWsMessageData<{ rsaKey: string }>) => {
  try {     const serverRsa = msg.data.rsaKey;
    setSessionRsa(serverRsa);  };
 const sendSessionAes = () => {
```

```
  try {     if (!sessionRsa) {return;     }
    const newSessionAesKey = aes.generateKey();
    const encryptedSessionAesKey = rsa.encryptData(
      sessionRsa,
      newSessionAesKey,
    );
    setSessionAes(newSessionAesKey);
    wsSend({
      event: "auth.set.aes",
      data: encryptedSessionAesKey,
    });
 };
 const sendReg = (name: string, password: string) => {
  const reqData = {
    name,
    password,
    rsaKey: "",
  };
```

Також необхідним є реалізація функція яка буде шифрувати повідомлення яке відправляється на сервер:

```
  const encryptData = ssl.encrypt(sessionAes, reqData);
  wsSend({
    event: "auth.registration",
    data: encryptData.data,
    iv: encryptData.iv,    });  }
```

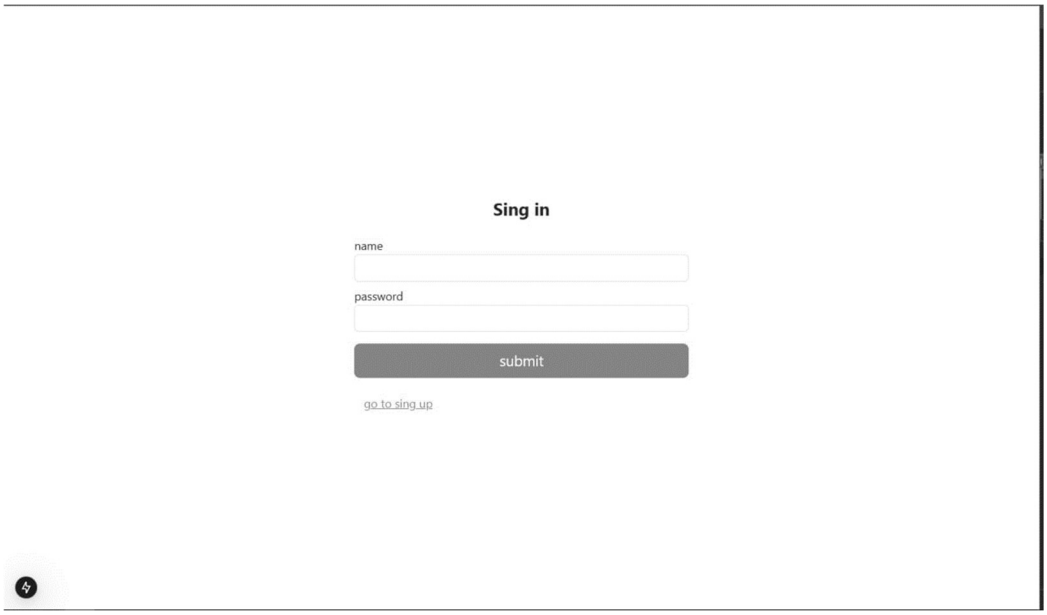

Рисунок 3.2.1 – Зображення сторінки авторизації користувача

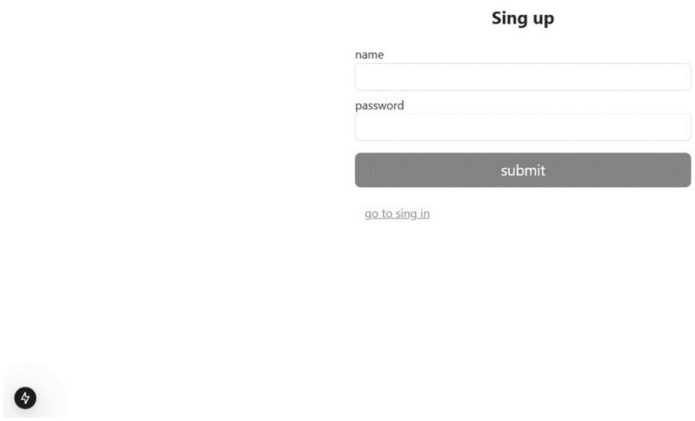

Рисунок 3.2.2 – Зображення сторінки реєстрації користувача

Після реалізації можливості авторизуватись необхідно виконати з’єднання з вервером:

```
const wsConnection = useRef<WebSocket | null>(null);
const wsListeners = useRef<IWsListeners<any>[]>([]); // eslint-disable-line
const [isConnect, setIsConnect] = useState(false);
useEffect(() => {
  if (!process.env.NEXT_PUBLIC_WS) {
    return;
  }
  wsConnection.current = new WebSocket(process.env.NEXT_PUBLIC_WS);
  wsConnection.current.onopen = () => {
    setIsConnect(true);
  };
  wsConnection.current.onclose = () => {
    setIsConnect(false);
  };
  wsConnection.current.onmessage = event => {
    handleMessage(event.data);
  };
  return () => {
    wsConnection.current?.close();
  };
}, []);
```

За допомогою даної функції відбувається обробка повідомлень які приходять клієнту від серверу по протоколу WebSocket:

```
const handleMessage = (data: string) => {
  try {
    // eslint-disable-next-line
    const parsedData = JSON.parse(data) as IWsMessageData<any>;
    wsListeners.current.forEach(({ event, cb }) => {
      if (event === parsedData.event) {
        cb(parsedData);
      }
    });
  } catch (error) {
    console.error(error);
  }
};
```

Дана функція створена аби в була можливість створити підписку на виконання певного коду при отриманні відповідних повідомлень від серверу:

```
const addWsListener = <T,>(event: string, cb: (msg: T) => void) => {
  wsListeners.current.push({ event, cb });
};
```

Дана функція видаляє створені підписки:

```
const removeWsListener = <T,>(event: string, cb: (msg: T) => void) => {
  wsListeners.current = wsListeners.current.filter(
    l => !(l.event === event && l.cb === cb),
  );
};
const wsSend = <T,>(data: T) => {
  if (!isConnect) {
    return;
  }
  wsConnection.current?.send(JSON.stringify(data));
};
```

Наступним кроком є реалізація шифрування та дешифрування повідомлень, оскільки трафік який передається в мережі є зашифрованим.

```
const [chat, setChat] = useState<IChat | undefined>(undefined);
const { isConnect, addWsListener, removeWsListener, wsSend } =
  useWebSocketContext();
const { sessionAes } = useAuthContext();
const { setChats } = useChatsContext();
useEffect(() => {
  addWsListener("chat.add.res", createChatRes);
  addWsListener("chat.getById.res", getChatRes);
  return () => {
    removeWsListener("chat.add.res", createChatRes);
    removeWsListener("chat.getById.res", getChatRes);
```

```
  };
 }, [, sessionAes]);
```

Дана функція дозволяє створити новий чат та перед відправкою шифрує повідомлення за допомогою симетричного алгоритму шифрування:

```
const createChatRes = (msg: IWsMessageData<any>) => {
 if (!sessionAes) {
  return;
 }
 const res = ssl.decrypt<IChat>(sessionAes, msg.data);
 setChats(prev => [...prev, { id: res.id, name: res.chatName }]);
};
const getChatRes = (msg: IWsMessageData<any>) => {
 if (!sessionAes) {
  return;
 }
 const res = ssl.decrypt<IChat>(sessionAes, msg.data);
 setChat(res);
};
const createChatReq = (chatName: string, clientId: string) => {
 const req = {
  chatName,
  clientId,
  keyOwner: aes.generateKey(),
  keyClient: aes.generateKey(),
 };
 const encryptData = aes.encryptData(sessionAes, JSON.stringify(req));
 wsSend({
  event: "chat.add",
  iv: encryptData.iv,
  data: encryptData.data,
 });
};
```

Дана функція отримає від сервера перелік чатів користувача та дешифрує за допомогою симетричного алгоритму шифрування:

```
const getChatReq = (id: string) => {
 const req = {
  chatId: id,  };  const encryptData = aes.encryptData(sessionAes, JSON.stringify(req));
 wsSend({    event: "chat.getById",
  iv: encryptData.iv,
  data: encryptData.data,
 });};
```

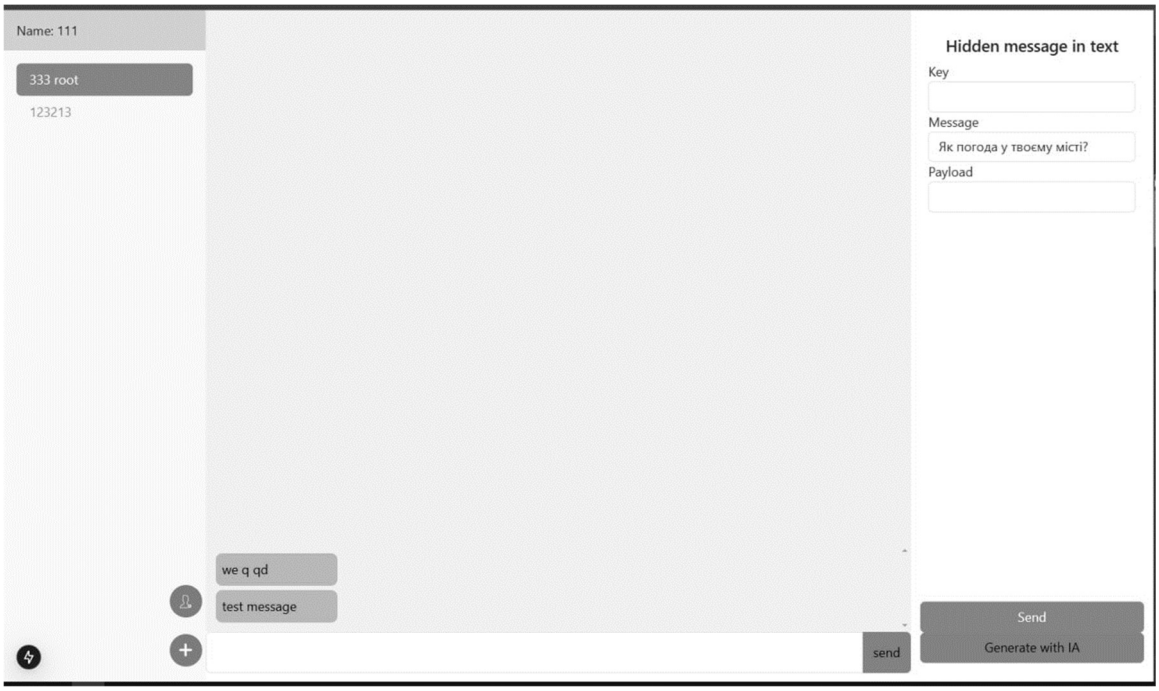

Рисунок 3.1.2 – знімок застосунку з активним користувацьким чатом

В даному підрозділі було розглянуто розробку програмного коду для реалізації клієнтської частини додатку спрямованого на конфіденційну передачу інформації. В наступному підрозділі буде розглянуто реалізацію серверної частини додатку.

3.2 Розробка серверної частини

Для програмної реалізації серверної частини буде використано класичний підхід Model View Controller з адаптацією під використання з WebSocket, оскільки всі контролери мають викликатись в середині обробника WebSocket.

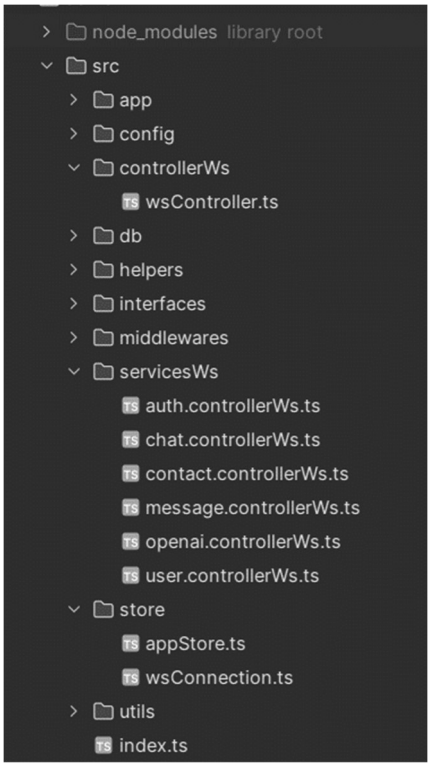

Рисунок 3.2.1 – зображення архітектури серверного додатку

Для початку реалізації серверної логіки необхідно розгорнути сервер:

```
const modules = [
  cors(),
  express.json({
    type: ["application/json"],
    limit: "1mb",
  }),
  express.urlencoded({ extended: true }),
  notFoundMiddleware,
  errorMiddleware,
];
modules.forEach((elm: any) => app.use(elm));
pgSqlConnect();
```

Даний код відповідає за розгортання HTTP серверу який буде використовуватись для побудови WebSocket з’єднання

```
export const httpServer = app.listen(PORT, () => {
  console.log(`=== Started http server port(${PORT})...`);
});
export const wsServer = new WebSocket.Server({
  noServer: true,
});
wsController();
```

```
upgradeHttpToWs();
appStore.init()
```

Даний код відповідає за оновлення http з’єднання та переведення його на WebSocket. Даний алгоритм виконується для запуску лише одного серверу на одному порту але який зможе приймати повідомлення за різними протоколами обміну даних

```
httpServer.on("upgrade", async function upgrade(request, socket, head) {
  if (request.url !== "/ws") {
    socket.destroy();
    return;
  }
  wsServer.handleUpgrade(request, socket, head, function done(ws) {
    wsServer.emit("connection", ws, request);
  });
});
```

Даний код відповідає за обробку повідомлення та відповідно до назви повідомлення розподіляє обробники:

```
wsServer.on("connection", async function (ws) {
 addConnectionWs(ws);
 ws.on("message", async function (data) {
   const msg = JSON.parse(data.toString());
   if (msg.event === routes.auth.rsa) {
     await authRsaControllerWs({ ws, msg });
     return;    }    if (msg.event === routes.auth.aes) {
     await authSetAesControllerWs({ ws, msg: msg.data });
     return;    }    if (msg.event === routes.auth.login) {
     await authLoginControllerWs({ ws, msg });
     return;    }    if (msg.event === routes.auth.registration) {
     await authRegistrationControllerWs({ ws, msg });
     return;
   }    if (msg.event === routes.user.getById) {
     await getUserControllerWs({ ws, msg });
     return;
   }    if (msg.event === routes.chat.getAll) {
     await getChatListControllerWs({ ws, msg });
     return;    }    });
 ws.on("close", function () {
   deleteConnectionWs(ws);
 });
});
```

Вигляд функції обробки повідомлення при початковому з’єднанні та початку обміну ключами асиметричного алгоритму шифрування:

```
export const authRsaControllerWs = async ({ ws, msg }: IWsController<unknown>) => {
 wsSend(ws, responses.auth.rsa, {
  rsaKey: appStore.get().rsa.public,
 });
};
```

Даний код реалізує збереження користувацького ключа від симетричного алгоритму шифрування для передачі даних в мережі:

```
export const authSetAesControllerWs = async ({ ws, msg }: IWsController<string>) => {
 const wsConnection = getConnectionWs(ws);
 if (!wsConnection) {
  return;
 }
 const msgData = rsa.decryptData(appStore.get().rsa.private, msg) as string;
 updateConnectionWs({ ws, aesKey: msgData });
 wsSend(ws, responses.auth.aes);
};
```

Даний код реалізує авторизацію користувача після встановлення з’єднання та обміну ключами сесії:

```
export const authLoginControllerWs = async ({ ws, msg }: IWsController<IWsControllerData>) => {
 const wsConnection = getConnectionWs(ws);
 if (!wsConnection) {    return;  }
 const msgData = JSON.parse(aes.decryptData(wsConnection.aesKey, msg.iv, msg.data)) as
IAuthLoginControllerWs;
 const findUserByLogin = await pgSql<IPgUser, true>(
  `SELECT *     FROM ${userTableName}     WHERE name = $1`,
  [msgData.name],
  true,
 );
 if (!findUserByLogin) {    return;  }
 const verify = await bcrypt.compare(msgData.password, findUserByLogin.password);
 if (!verify) {
  return;
 }
 updateConnectionWs({ ws, isAuth: true, userId: findUserByLogin.id });
 const userRsaKeys = await pgSql<IPgUserRsaKey, true>(
  `SELECT *
   FROM ${userRsaKeyTableName}
   WHERE user_id = $1
    AND is_main = $2`,    [findUserByLogin.id, true],    true,
 );
```

```
  const result = aes.encryptData(
    wsConnection.aesKey,
    JSON.stringify({
      id: findUserByLogin.id,
      name: findUserByLogin.name,
      rsaKey: "userRsaKeys.user_key",
    }),
  );
  wsSend(ws, responses.auth.login, {
    iv: result.iv,
    data: result.data,
  });
};
```

В даному підрозділі було розглянуто процес побудови серверної частини додатку, розглянуто його архітектуру а також реалізовано основні його частини, серед який є функції з отримання та обміну ключами сесії які використовуються під час з’єднання клієнта і сервера та функції авторизації користувача.

3.3 Розгортання інфраструктури для хостингу додатку

Для розгортання інфраструктури додатку буде використано Amazon Web Services який надає доступ до хмарної інфраструктури та забезпечує велику стабільність роботи та безпеку даних.

Для розгортання додатку буде використовуватись патерн синій – зелений деплой, що в свою чергу дозволяє безпечно оновлювати версії застосунків. При використанні даного процесу доставки додатків необхідно мати дві сервери де на кожному розгорнуто по одному серверу, в разі відмови або оновлення одного з серверів, весь трафік користувачів перемикається на інший та забезпечує роботу додатку коли інший сервер тимчасово недоступний. Після процесу оновлення в розробників також є можливість протестувати сервер перед тим як пустити на нього основний трафік користувачів та в разі його відмови, користувачів можна повернути назад на старий сервер.

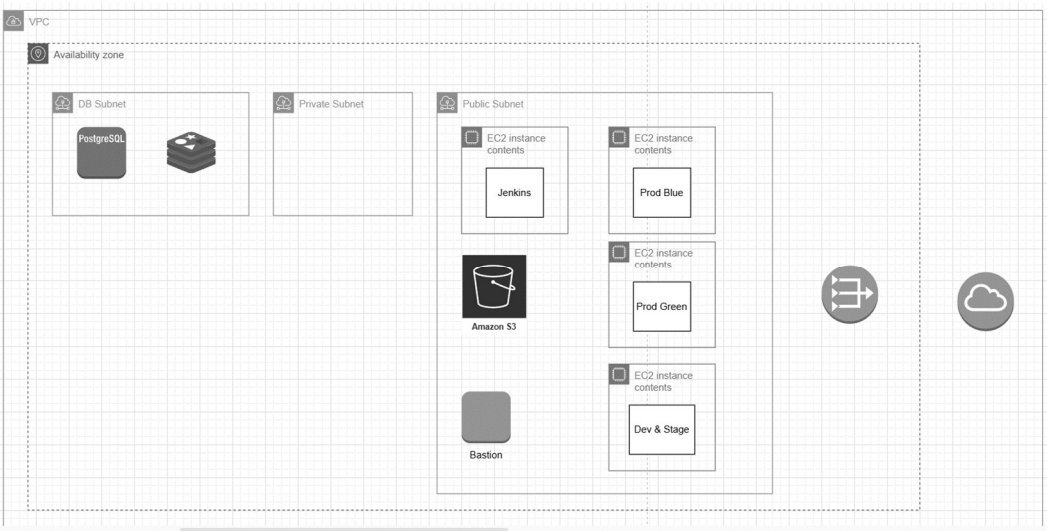

Рисунок 3.3.1 – схема архітектури додатку

На рисунку 3.3.1 можна побачити віртуальну приватну мережу в якій знаходить одна зона доступу яку надає AWS, в зоні в свою чергу знаходиться при підмережі серед яких є мережа баз даних, приватна мережа та публічна.

В мережі баз даних знаходяться бази даних та до них можна отримати доступ тільки з середини віртуальної приватної мережі, що в свою чергу забезпечує суттєвий захист бази даних та унеможливлює отримання доступу зовні мережі.

Приватна мережа створена для можливого розширення застосунку а саме при розподіленні його на мікросервіси.

Публічна мережа яка має повний доступ в інтернет містить в собі сервери додатків, Bastion який дозволяє підключитись до бази даних при необхідності та сховища.

Для доступу в загальну мережу використовується NAT за допомогою якого публічна та приватна підмережа може мати доступ в інтернет.

3.4 Тестування застосунку

Для початку використання додатку, необхідно виконати його тестування. Для використання додатку необхідно мати веб браузер який підтримує JavaScript та може його виконувати.

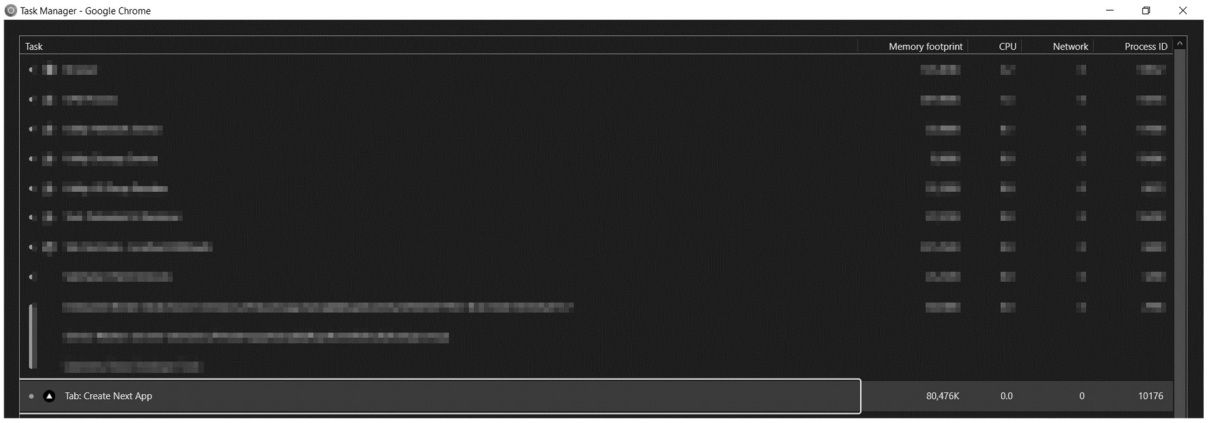

Рисунок 3.4.1 – диспетчер завдань браузера chrome

На рисунку 3.4.1 можна побачити кількість ресурсів яке споживає активна вкладка браузеру з відкритим застосунком. Дані які показані на рисунку, а саме кількість споживаної пам’яті 80.4 кілобайти свідчать про те, що застосунок можна використовувати майже на любому слабкому пристрої.

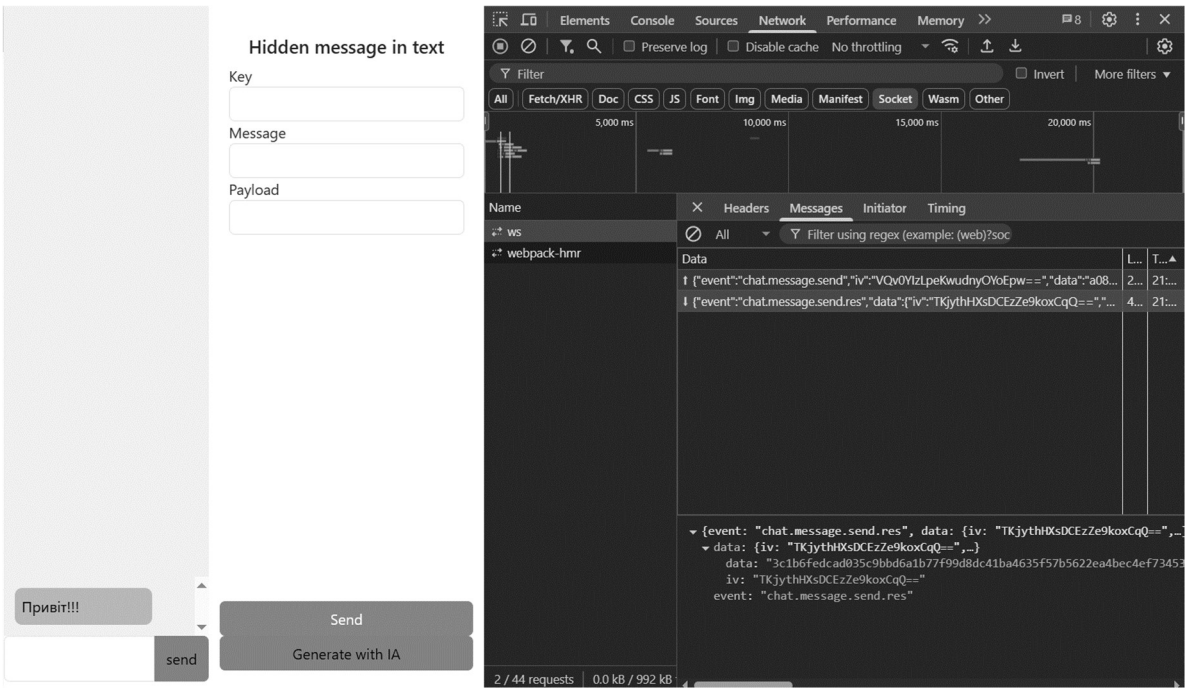

Рисунок 3.4.2 – знімок екрану під час відправки повідомлення

На рисунку 3.4.2 зображено вміст переданого повідомлення, і дані знаходяться в зашифрованому вигляді.

Для перевірки змісту повідомлення необхідно виконати його відправку та перевірити вміст а також на стороні співрозмовника перевірити чи дані отримані правильно як саме повідомлення так і прихований мвіст.

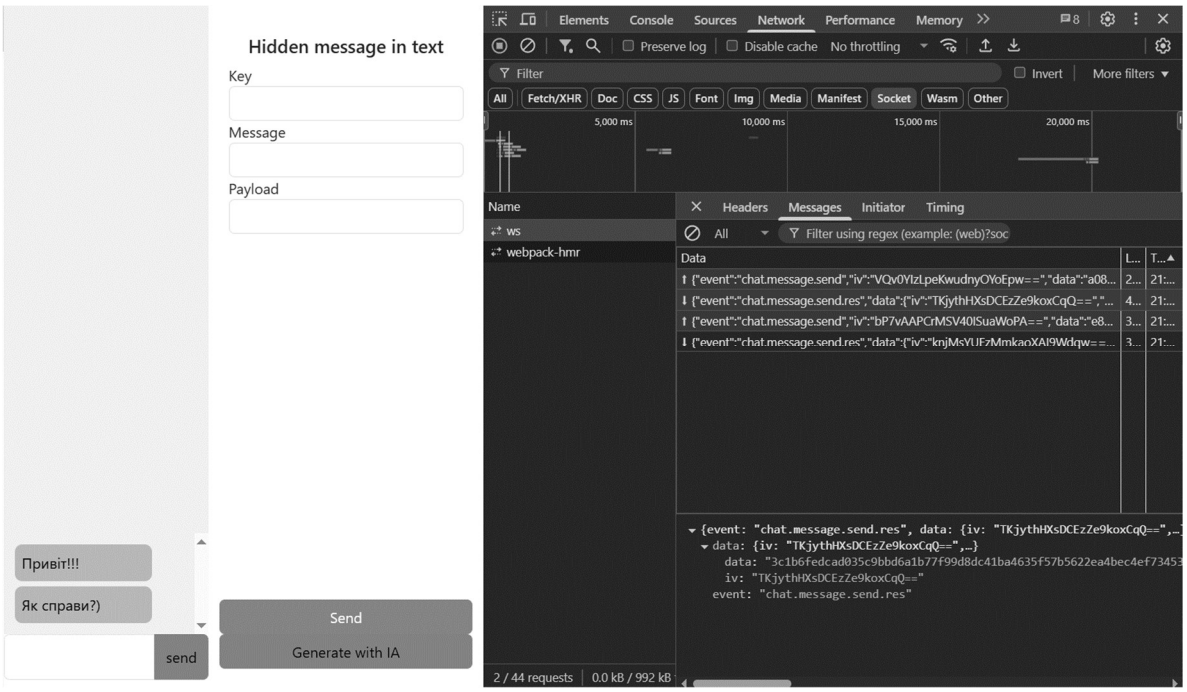

Рисунок 3.4.3 – знімок екрану під час передачі повідомлення з прихованим вмістом

Можна побачити, що повідомлення яке відправлено з прихованим вмістом ніяк не відрізняється від звичайного текстового повідомлення.

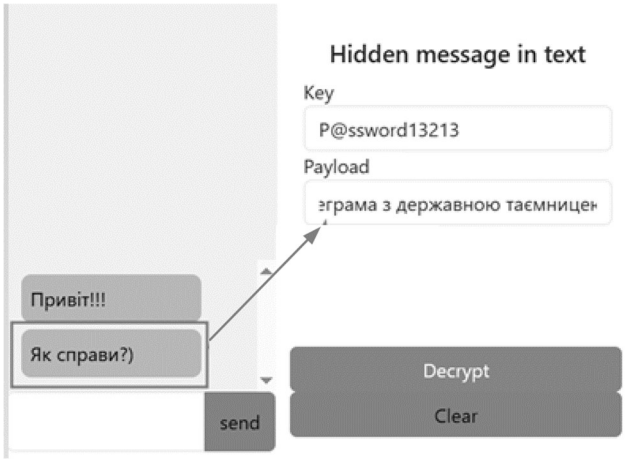

Рисунок 3.4.4 – знімок екрану з демонстрацією прихованого вмісту повідомлення

Після введення ключа приховане повідомлення правильно відображається в полі payload та це свідчить про коректну передачу повідомлення та його прихованого вмісту.

3.5 Висновки

В даному розділі було розглянуто програмну реалізації серверної та клієнтської частини месенджера для конфеденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, технологій стеганографії та моделей штучного інтелекту для маскування прихованих даних. Було побудовано та розглянуто архітектуру додатку яка базується на використанні хмарних технологій AWS.

Було реалізовано обмін повідомленнями між серверною та клієнтською частиною з можливістю шифрування та дешифрування трафіку який може передаватись по незахищеному каналу зв’язку. Був реалізований процес обміну ключами для використання криптографічних методів захисту інформації.

Також враховані можливі атаки на базу даних, оскільки вона знаходиться в підмережі яка не має доступу в інтернет, користувачі які не мають доступу до віртуальної приватної мережі не зможуть підключитись до бази даних та отримати інформацію.

ВИСНОВКИ

В даній бакалаврській роботі було розглянуто тему розробки месенджера для конфеденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, технологій стеганографії та моделей штучного інтелекту для маскування прихованих даних.

Під час вивчення предметної області було обрано алгоритм Stegcloak який дозволяє виконувати приховування даних в тексті завдяки невидимим символам які використовуються як відступи між словами. Було виконано інтеграцію з штучним інтелектом від OpenAI за допомогою якого виконується генерація тексту який потім може використовуватись для маскування іншого повідомлення користувача. Також було використано AES та RSA алгоритми для побудови безпечного каналу зв’язку

Для програмної реалізації було обрано сучасні бібліотеки та фреймворки, які дозволяють швидко і легко виконувати розробку додатку. Було реалізовано основний функціонал додатку який включає в себе можливість передавати зашифровані повідомлення між користувачами, можливість використати стеганографічні методи приховування інформації в тексті за допомогою алгоритму Stegcloak та можливість згенерувати повідомлення за допомогою AI. Також в користувача є можливість створювати нові чати, додавати нових контактів та дешифрувати приховані повідомлення.

Дана розробка є потужним застосунком для обміну даними з можливістю використовувати в різних задачах, серед яких може бути як передача секретних відомостей, координація або інформування людей так і передача паролей.

Використання стеганографії дозволяє реалізувати додатковий шар захисту, та може захистити від атак, підглядання через плече а також додатково шифрує повідомлення, оскільки stegclock використовує шифрування за допомгою AES.

ПЕРЕЛІК ПОСИЛАНЬ

1. ДСТУ ISO/IEC 18033-3:2015 Інформаційні технології. Методи захисту. Алгоритми шифрування. Частина 3. Блокові шифри (ISO/IEC 18033-3:2010, IDT). URL: https://online.budstandart.com/ua/catalog/doc-page?id_doc=65060 (дата звернення: 02.04.2025).

2. *NIST Technical Series Publications*. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (дата звернення: 02.04.2025).

3. What Is AES? How Does It Work? | Encryption Consulting. *Encryption Consulting*. URL: https://www.encryptionconsulting.com/education-center/what-is-aes/ (дата звернення: 02.04.2025).

4. AES Encryption: What is it & How Does it Safeguard your Data?. *Network security platform for business | NordLayer*. URL: https://nordlayer.com/blog/aes-encryption/ (дата звернення: 02.04.2025).

5. RSA algorithm structure. *https://www.researchgate.net*. URL: https://www.researchgate.net/figure/RSA-algorithm-structure_fig2_298298027 (дата звернення: 02.04.2025).

6. Алгоритм шифрування RSA, види атак на нього. URL: https://dou.ua/forums/topic/43026/ (дата звернення: 04.04.2025).

7. Exploring E2EE: Real-world Examples of End-to-End Encryption. *Kiteworks | Your Private Data Network*. URL: https://www.kiteworks.com/secure-file-sharing/real-world-examples-of-end-to-end-encryption/#How_Strong_Is_End-to-End_Encryption (дата звернення: 04.04.2025).

8. GeeksforGeeks. End to End Encryption (E2EE) in Computer Networks - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/end-to-end-encryption-e2ee-in-computer-networks/ (дата звернення: 08.04.2025).

9. Unlocking the Secrets of Steganography in Cybersecurity. *Wizard Cyber*. URL: https://wizardcyber.com/unlocking-the-secrets-of-steganography-in-cybersecurity/ (дата звернення: 08.04.2025).

10. Що таке принцип Керкгоффса – Терміни та визначення у сфері кібербезпеки. *VPN Unlimited - Fast & Secure VPN service*. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/kerckhoffs-principle (дата звернення: 08.04.2025).

11. Mohanasundar. How to Hide Secrets in Strings– Modern Text hiding in JavaScript. *Medium*. URL: https://blog.bitsrc.io/how-to-hide-secrets-in-strings-modern-text-hiding-in-javascript-613a9faa5787 (дата звернення: 08.04.2025).

12. Жолукевський. *КОНФЕРЕНЦІЇ ВНТУ електронні наукові видання*. URL: https://conferences.vntu.edu.ua/index.php/all-fbtegp/all-fbtegp-2023/paper/view/17344/14477 (дата звернення: 06.04.2025).

13. What Is Artificial Intelligence (AI)? | Built In. *Built In*. URL: https://builtin.com/artificial-intelligence (дата звернення: 02.04.2025).

14. AI: A Complete Guide in Simple Terms. *WEKA*. URL: https://www.weka.io/learn/guide/ai-ml/what-is-ai/ (дата звернення: 07.04.2025).

15. K R. Як працює ChatGPT простими словами. *Друкарня*. URL: https://drukarnia.com.ua/articles/yak-pracyuye-chatgpt-prostimi-slovami-MtpBS#heading-2-965 (дата звернення: 06.04.2025).

16. *Red Hat - We make open source technologies for the enterprise*. URL: https://www.redhat.com/rhdc/managed-files/blue-green-deployment-model.gif (дата звернення: 09.04.2025).

17. What is blue green deployment?. *Red Hat - We make open source technologies for the enterprise*. URL: https://www.redhat.com/en/topics/devops/what-is-blue-green-deployment (дата звернення: 09.04.2025).

18. What is WebSocket and How It Works?. *Wallarm | Advanced API Security*. URL: https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is (дата звернення: 10.04.2025).

19. HTTP request methods - HTTP | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (дата звернення: 10.04.2025).

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ

Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор

Юрій ЯРЕМЧУК

" 20 " березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської кваліфікаційної роботи на тему:

Розробка месенджера для конфеденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, технологій стеганографії та моделей штучного інтелекту для маскування прихованих даних

08-72.БКР.005.00.052.ТЗ

Керівник бакалаврської кваліфікаційної роботи

к.т.н., доцент каф. МБІС

Грицак А.В.

" 20 " березня 2025 р.

Вінниця – 2025 р.

1. Найменування та область застосування

Розробка месенджера для конфеденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, технологій стеганографії та моделей штучного інтелекту для маскування прихованих даних. Область застосування: захист конфіденційної інформації під час передачі повідомлень.

2.Підстава для розробки

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

3. Мета та призначення розробки

3.1 Мета розробки: розробка ефективного додатку для безпечної педерачі конфіденційної інформації.

3.2 Призначення: розроблений програмний засіб виконує передачу конфіденційної інформації по реалізованому захищеному каналу зв’язку.

4. Джерела розробки

4.1. RSA algorithm structure. https://www.researchgate.net. URL: https://www.researchgate.net/figure/RSA-algorithm-structure_fig2_298298027.

4.2. AES Encryption: Study & Evaluation URL: https://www.researchgate.net/publication/346446212_AES_Encryption_Study_Evaluation.

4.3. A Whitespace Replacement Information Hidingmethod. URL: https://www.researchgate.net/publication/389130097_TREND_A_Whitespace_Replacement_Information_Hiding_Method.

4.4. Generative AI-Based Text Generation Methods Using Pre-Trained GPT-2 Model. URL: https://www.researchgate.net/publication/379511447_Generative_AI-Based_Text_Generation_Methods_Using_Pre-Trained_GPT-2_Model.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен виконувати процес автентифікації користувачів у системі.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Бази даних повинні бути налаштовані на автоматичне створення резервних копій;

5.2.3 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

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

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

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

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

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	22.03.2025	24.03.2025
2.	Аналіз предметної області обраної теми	25.03.2025	26.03.2025
3.	Розробка алгоритму роботи	30.03.2025	10.04.2025
4.	Написання бакалаврської роботи на основі розробленої теми	12.04.2025	20.04.2025
5.	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025
6.	Виправлення, уточнення, корегування бакалаврської кваліфікаційної роботи	28.05.2025	01.06.2025
7.	Захист бакалаврської кваліфікаційної роботи	11.06.2025	12.06.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

– ПЗ до бакалаврської кваліфікаційної роботи;

– демонстрація результату бакалаврської кваліфікаційної роботи;

– презентація;

– відзив керівника роботи;

– відзив рецензента

Технічне завдання до виконання прийняв Жолукевський В.В.

Додаток Б. Лістинг

```
import dotenv from "dotenv";
import express from "express";
import cors from "cors";
import WebSocket from "ws";
dotenv.config();
const app = express();
const PORT = 3001;
import { errorMiddleware } from "./middlewares/error.middleware";
import { pgSqlConnect } from "./db/pgSqlConnect";
import { wsController } from "./controllerWs/wsController";
import { upgradeHttpToWs } from "./helpers/upgradeHttpToWs";
import { notFoundMiddleware } from "./middlewares/notFoundMiddleware";
import { appStore } from "./store/appStore";
import OpenAI from "openai";
import * as process from "node:process";
const modules = [
  cors(),
  express.json({
    type: ["application/json"],
    limit: "1mb",
  }),
  express.urlencoded({ extended: true }),
  notFoundMiddleware,
  errorMiddleware,
];
modules.forEach((elm: any) => app.use(elm));
pgSqlConnect();
export const openai = new OpenAI({
  organization: process.env.GPT,
  apiKey: "",
});
export const httpServer = app.listen(PORT, () => {
  console.log(`=== Started http server port(${PORT})...`);
});
export const wsServer = new WebSocket.Server({
  noServer: true,
});
wsController();
upgradeHttpToWs();
appStore.init()
import { NextFunction, Request, Response } from "express";
export const errorMiddleware = (
  err: Error,
  req: Request,
  res: Response,
  next: NextFunction,
) => {
```

```
  console.error("!!! ERROR !!!", err.message);
  return res.status(500).json({ error: "server error" });
};
import { Request, Response, Router } from "express";
const notFoundController = (req: Request, res: Response) => {
  return res.status(404).json({ error: "404" });
};
const notFound = new Router();
notFound.get(`/*`, notFoundController);
notFound.post(`/*`, notFoundController);
notFound.put(`/*`, notFoundController);
notFound.delete(`/*`, notFoundController);
export const notFoundMiddleware = notFound;
import { pgSql } from "../db/pgSql";
import { userRsaKeyTableName, userTableName } from "../db/pgTables";
import * as bcrypt from "bcrypt";
import { getConnectionWs, updateConnectionWs } from "../store/wsConnection";
import { IWsController, IWsControllerData } from "../interfaces/ws/wsController";
import { responses } from "../config/responses";
import { appStore } from "../store/appStore";
import { wsSend } from "../helpers/wsSend";
import { aes } from "../utils/aes";
import { rsa } from "../utils/rsa";
export const authRsaControllerWs = async ({ ws, msg }: IWsController<unknown>) => {
  wsSend(ws, responses.auth.rsa, {
    rsaKey: appStore.get().rsa.public,
  });
};
export const authSetAesControllerWs = async ({ ws, msg }: IWsController<string>) => {
  const wsConnection = getConnectionWs(ws);
  if (!wsConnection) {
    return;
  }
  const msgData = rsa.decryptData(appStore.get().rsa.private, msg) as string;
  updateConnectionWs({ ws, aesKey: msgData });
  wsSend(ws, responses.auth.aes);
};
export const authLoginControllerWs = async ({ ws, msg }: IWsController<IWsControllerData>)
=> {
  const wsConnection = getConnectionWs(ws);
  if (!wsConnection) {
    return;
  }
  const msgData = JSON.parse(aes.decryptData(wsConnection.aesKey, msg.iv, msg.data)) as
IAuthLoginControllerWs;
  const findUserByLogin = await pgSql<IPgUser, true>(
    `SELECT *
     FROM ${userTableName}
     WHERE name = $1`,
```

```
    [msgData.name],
    true,
  );
  if (!findUserByLogin) {
    return;
  }
  const verify = await bcrypt.compare(msgData.password, findUserByLogin.password);
  if (!verify) {
    return;
  }
  updateConnectionWs({ ws, isAuth: true, userId: findUserByLogin.id });
  const userRsaKeys = await pgSql<IPgUserRsaKey, true>(
    `SELECT *
     FROM ${userRsaKeyTableName}
     WHERE user_id = $1
       AND is_main = $2`,
    [findUserByLogin.id, true],
    true,
  );
  const result = aes.encryptData(
    wsConnection.aesKey,
    JSON.stringify({
      id: findUserByLogin.id,
      name: findUserByLogin.name,
      rsaKey: "userRsaKeys.user_key",
    }),
  );
  wsSend(ws, responses.auth.login, {
    iv: result.iv,
    data: result.data,
  });
};
export const authRegistrationControllerWs = async ({ ws, msg }:
IWsController<IWsControllerData>) => {
  const wsConnection = getConnectionWs(ws);
  if (!wsConnection) {
    return;
  }
  const msgData = JSON.parse(aes.decryptData(wsConnection.aesKey, msg.iv, msg.data)) as
IAuthRegistrationControllerWs;
  const findUserInDb = await pgSql<IPgUser, true>(
    `SELECT *
     FROM ${userTableName}
     WHERE name = $1`,
    [msgData.name],
    true,
  );
  if (findUserInDb) {
    return;
```

```
  }
  const passwordHash = await bcrypt.hash(msgData.password, 10);
  await pgSql(
    `INSERT INTO ${userTableName} (name, password)
     VALUES ($1, $2)`,
    [msgData.name, passwordHash],
  );
  wsSend(ws, responses.auth.registration);
};
import { IWsController, IWsControllerData } from "../interfaces/ws/wsController";
import { wsSend } from "../helpers/wsSend";
import { responses } from "../config/responses";
import { getConnectionWs, getWsByUserId } from "../store/wsConnection";
import { aes } from "../utils/aes";
import { pgSql } from "../db/pgSql";
import { chatTableName, chatToUserTableName, userChatKeyTableName, userTableName }
from "../db/pgTables";
export const getChatListControllerWs = async ({ ws, msg }: IWsController<IWsControllerData>)
=> {
  const wsConnection = getConnectionWs(ws);
  if (!wsConnection || !wsConnection.isAuth) {
    return;
  }
  const getChatsToUser = await pgSql<IPgChatToUser, false>(
    `SELECT *
     FROM ${chatToUserTableName}
     WHERE user_id = $1`,
    [wsConnection.userId],
    false,
  );
  const userChats = [];
  for (let i = 0; i < getChatsToUser.length; i++) {
    const getChat = await pgSql<IPgChat, true>(
      `SELECT *
       FROM ${chatTableName}
       WHERE id = $1`,
      [getChatsToUser[i].chat_id],
      true,
    );
    userChats.push({
      id: getChat.id,
      name: getChat.name,
    });
  }
  wsSend(ws, responses.chat.getAll, aes.encryptData(wsConnection.aesKey,
JSON.stringify(userChats)));
};
export const getChatByIdControllerWs = async ({ ws, msg }:
IWsController<IWsControllerData>) => {
```

```
  const wsConnection = getConnectionWs(ws);
  if (!wsConnection || !wsConnection.isAuth) {
    return;
  }
  const msgData = JSON.parse(aes.decryptData(wsConnection.aesKey, msg.iv, msg.data)) as
any;
  const getChat = await pgSql<IPgChat, true>(
    `SELECT *
     FROM ${chatTableName}
     WHERE id = $1`,
    [msgData.chatId],
    true,
  );
  const getChatUserClient = await pgSql<IPgChatToUser, true>(
    `SELECT *
     FROM ${chatToUserTableName}
     WHERE chat_id = $1
       AND user_id != $2`,
    [msgData.chatId, wsConnection.userId],
    true,
  );
  if (!getChatUserClient) {
    return;
  }
  const getClientData = await pgSql<IPgUser, true>(
    `SELECT *
     FROM ${userTableName}
     WHERE id = $1`,
    [getChatUserClient.user_id],
    true,
  );
  const userChatKey = await pgSql<IPgUserChatKey, true>(
    `SELECT *
     FROM ${userChatKeyTableName}
     WHERE user_id = $1
       AND chat_id = $2`,
    [wsConnection.userId, msgData.chatId],
    true,
  );
  wsSend(
    ws,
    responses.chat.getById,
    aes.encryptData(
      wsConnection.aesKey,
      JSON.stringify({
        id: msgData.chatId,
        chatName: getChat.name,
        clientId: getChatUserClient.id,
        clientName: getClientData.name,
```

```
      aesKey: userChatKey.aes_key,
     }),
    ),
  );
};
export const addChatControllerWs = async ({ ws, msg }: IWsController<IWsControllerData>) =>
{
  const wsConnection = getConnectionWs(ws);
  if (!wsConnection || !wsConnection.isAuth) {
    return;
  }
  const msgData = JSON.parse(aes.decryptData(wsConnection.aesKey, msg.iv, msg.data)) as
any;
  console.log(msgData);
  const findClientByName = await pgSql<IPgUser, true>(
    `SELECT *
     FROM ${userTableName}
     WHERE id = $1`,
    [msgData.clientId],
    true,
  );
  console.log(findClientByName);
  console.log(2);
  const findOwner = await pgSql<IPgUser, true>(
    `SELECT *
     FROM ${userTableName}
     WHERE id = $1`,
    [wsConnection.userId],
    true,
  );
  if (!findClientByName) {
    return;
  }
  const newChat = await pgSql<IPgChat, true>(
    `INSERT INTO ${chatTableName} (name)
     VALUES ($1) RETURNING *`,
    [msgData.chatName],
    true,
  );
  console.log(4);
  const newChatToUserOwner = await pgSql<IPgChatToUser, true>(
    `INSERT INTO ${chatToUserTableName} (user_id, chat_id)
     VALUES ($1, $2) RETURNING *`,
    [wsConnection.userId, newChat.id],
    true,
  );
  const newChatToUserClient = await pgSql<IPgChatToUser, true>(
    `INSERT INTO ${chatToUserTableName} (user_id, chat_id)
     VALUES ($1, $2) RETURNING *`,
```

```
  [findClientByName.id, newChat.id],
  true,
);
const newChatToUserKeyOwner = await pgSql<IPgChatToUser, true>(
  `INSERT INTO ${userChatKeyTableName} (user_id, chat_id, aes_key)
   VALUES ($1, $2, $3) RETURNING *`,
  [wsConnection.userId, newChat.id, msgData.keyOwner],
  true,
);
const newChatToUserKeyClient = await pgSql<IPgChatToUser, true>(
  `INSERT INTO ${userChatKeyTableName} (user_id, chat_id, aes_key)
   VALUES ($1, $2, $3) RETURNING *`,
  [findClientByName.id, newChat.id, msgData.keyClient],
  true,
);
console.log(5);
console.log(aes.encryptData(
  wsConnection.aesKey,
  JSON.stringify({
    id: newChat.id,
    chatName: msgData.chatName,
    clientId: msgData.clientId,
    clientName: findClientByName.name,
    aesKey: msgData.keyOwner,
  }),
),);
wsSend(
  ws,
  responses.chat.add,
  aes.encryptData(
    wsConnection.aesKey,
    JSON.stringify({
      id: newChat.id,
      chatName: msgData.chatName,
      clientId: msgData.clientId,
      clientName: findClientByName.name,
      aesKey: msgData.keyOwner,
    }),
  ),
);
const clientWs = getWsByUserId(findClientByName.id);
if (!clientWs) {
  return;
}
console.log(6);
const clientConnectionWs = getConnectionWs(clientWs);
wsSend(
  clientWs,
  responses.chat.add,
```

```
    aes.encryptData(
      clientConnectionWs.aesKey,
      JSON.stringify({
        id: newChat.id,
        chatName: msgData.chatName,
        clientId: wsConnection.userId,
        clientName: findOwner.name,
        aesKey: msgData.keyClient,
      }),
    ),
  );
};
import { IWsController, IWsControllerData } from "../interfaces/ws/wsController";
import { wsSend } from "../helpers/wsSend";
import { responses } from "../config/responses";
import { getConnectionWs } from "../store/wsConnection";
import { aes } from "../utils/aes";
import { pgSql } from "../db/pgSql";
import { contactTableName, userTableName } from "../db/pgTables";
export const getContactListControllerWs = async ({ ws, msg }:
IWsController<IWsControllerData>) => {
  const wsConnection = getConnectionWs(ws);
  if (!wsConnection || !wsConnection.isAuth) {
    return;
  }
  const contactList = await pgSql<IPgContact, false>(
    `SELECT *
     FROM ${contactTableName}
     WHERE user_id = $1`,
    [wsConnection.userId],
    false,
  );
  const contactListRes = [];
  for (let i = 0; i < contactList.length; i++) {
    const findUserByName = await pgSql<IPgUser, true>(
      `SELECT *
       FROM ${userTableName}
       WHERE id = $1`,
      [contactList[i].contact_id],
      true,
    );
    contactListRes.push({ ...contactList[i], name: findUserByName.name });
  }
  wsSend(ws, responses.contact.getAll, aes.encryptData(wsConnection.aesKey,
JSON.stringify(contactListRes)));
};
export const addContactControllerWs = async ({ ws, msg }:
IWsController<IWsControllerData>) => {
  const wsConnection = getConnectionWs(ws);
```

```
  if (!wsConnection || !wsConnection.isAuth) {
    return;
  }
  const msgData = JSON.parse(aes.decryptData(wsConnection.aesKey, msg.iv, msg.data)) as
any;
  const findUserByName = await pgSql<IPgUser, true>(
    `SELECT *
     FROM ${userTableName}
     WHERE name = $1`,
    [msgData.name],
    true,
  );
  const newContact = await pgSql<IPgContact, true>(
    `INSERT INTO ${contactTableName} (contact_id, user_id)
     VALUES ($1, $2) RETURNING *`,
    [findUserByName.id, wsConnection.userId],
    true,
  );
  wsSend(
    ws,
    responses.contact.add,
    aes.encryptData(
      wsConnection.aesKey,
      JSON.stringify({
        newContact,
        name: msgData.name,
      }),
    ),
  );
};
import { IWsController, IWsControllerData } from "../interfaces/ws/wsController";
import { wsSend } from "../helpers/wsSend";
import { responses } from "../config/responses";
import { getConnectionWs, getWsByUserId } from "../store/wsConnection";
import { aes } from "../utils/aes";
import { pgSql } from "../db/pgSql";
import { chatToUserTableName, messageTableName } from "../db/pgTables";
export const getMessageListControllerWs = async ({ ws, msg }:
IWsController<IWsControllerData>) => {
  const wsConnection = getConnectionWs(ws);
  if (!wsConnection || !wsConnection.isAuth) {
    return;
  }
  const msgData = JSON.parse(aes.decryptData(wsConnection.aesKey, msg.iv, msg.data)) as
any;
  const chatMessages = await pgSql<IPgMessage, false>(
    `SELECT *
     FROM ${messageTableName}
     WHERE chat_id = $1`,
```

```
    [msgData.chatId],
    false,
  );
  wsSend(ws, responses.message.getAll, aes.encryptData(wsConnection.aesKey,
JSON.stringify(chatMessages)));
};
export const addMessageControllerWs = async ({ ws, msg }:
IWsController<IWsControllerData>) => {
  const wsConnection = getConnectionWs(ws);
  if (!wsConnection || !wsConnection.isAuth) {
    return;
  }
  const msgData = JSON.parse(aes.decryptData(wsConnection.aesKey, msg.iv, msg.data)) as
any;
  const saveMessage = await pgSql<IPgMessage, true>(
    `INSERT INTO ${messageTableName} (content, chat_id)
     VALUES ($1, $2) RETURNING *`,
    [msgData.content, msgData.chatId],
    true,
  );
  const findUserToChat = await pgSql<IPgChatToUser, true>(
    `SELECT *
     FROM ${chatToUserTableName}
     WHERE chat_id = $1
       AND user_id != $2`,
    [msgData.chatId, wsConnection.userId],
    true,
  );
  wsSend(ws, responses.message.send, aes.encryptData(wsConnection.aesKey,
JSON.stringify(saveMessage)));
  const clientWs = getWsByUserId(findUserToChat.user_id);
  if (!clientWs) {
    return;
  }
  const clientConnectionWs = getConnectionWs(clientWs);
  wsSend(clientWs, responses.message.send, aes.encryptData(clientConnectionWs.aesKey,
JSON.stringify(saveMessage)));
};
import { IWsController, IWsControllerData } from "../interfaces/ws/wsController";
import { wsSend } from "../helpers/wsSend";
import { responses } from "../config/responses";
import { getConnectionWs } from "../store/wsConnection";
import { aes } from "../utils/aes";
import { pgSql } from "../db/pgSql";
import { messageTableName, userRsaKeyTableName, userTableName } from "../db/pgTables";
export const getUserControllerWs = async ({ ws, msg }: IWsController<IWsControllerData>) =>
{
  const wsConnection = getConnectionWs(ws);
  if (!wsConnection || !wsConnection.isAuth) {
```

```
    return;
  }
  const msgData = JSON.parse(aes.decryptData(wsConnection.aesKey, msg.iv, msg.data)) as
any;
  const userData = await pgSql<IPgUser, true>(
    `SELECT *
     FROM ${userTableName}
     WHERE id = $1`,
    [msgData.userId],
    true,
  );
  const userRsaKeys = await pgSql<IPgUserRsaKey, true>(
    `SELECT *
     FROM ${userRsaKeyTableName}
     WHERE user_id = $1
       AND is_main = $2`,
    [msgData.userId, true],
    true,
  );
  wsSend(
    ws,
    responses.user.getById,
    aes.encryptData(
      wsConnection.aesKey,
      JSON.stringify({
        id: msgData.userId,
        name: userData.name,
        rsaKey: userRsaKeys.user_key,
      }),
    ),
  );
};
import { wsServer } from "../index";
import { addConnectionWs, deleteConnectionWs } from "../store/wsConnection";
import { routes } from "../config/routes";
import {  authLoginControllerWs,  authRegistrationControllerWs,  authRsaControllerWs,
  authSetAesControllerWs,
} from "../servicesWs/auth.controllerWs";
import { addChatControllerWs, getChatByIdControllerWs, getChatListControllerWs } from
"../servicesWs/chat.controllerWs";
import { addMessageControllerWs, getMessageListControllerWs } from
"../servicesWs/message.controllerWs";
import { addContactControllerWs, getContactListControllerWs } from
"../servicesWs/contact.controllerWs";
import { getUserControllerWs } from "../servicesWs/user.controllerWs";
import { openaiControllerWs } from "../servicesWs/openai.controllerWs";
export const wsController = () =>
  wsServer.on("connection", async function (ws) {
    addConnectionWs(ws);
```

```
ws.on("message", async function (data) {
  const msg = JSON.parse(data.toString());
  if (msg.event === routes.auth.rsa) {
    await authRsaControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.auth.aes) {
    await authSetAesControllerWs({ ws, msg: msg.data });
    return;     }
  if (msg.event === routes.auth.login) {
    await authLoginControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.auth.registration) {
    await authRegistrationControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.user.getById) {
    await getUserControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.chat.getAll) {
    await getChatListControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.chat.getById) {
    await getChatByIdControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.chat.add) {
    await addChatControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.message.getAll) {
    await getMessageListControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.message.send) {
    await addMessageControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.contact.getAll) {
    await getContactListControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.contact.add) {
    await addContactControllerWs({ ws, msg });
    return;     }
  if (msg.event === routes.openai) {
    await openaiControllerWs({ ws, msg });
    return;     }
});
ws.on("close", function () {
  deleteConnectionWs(ws);
});  });
```

Додаток В. Ілюстраційні матеріали

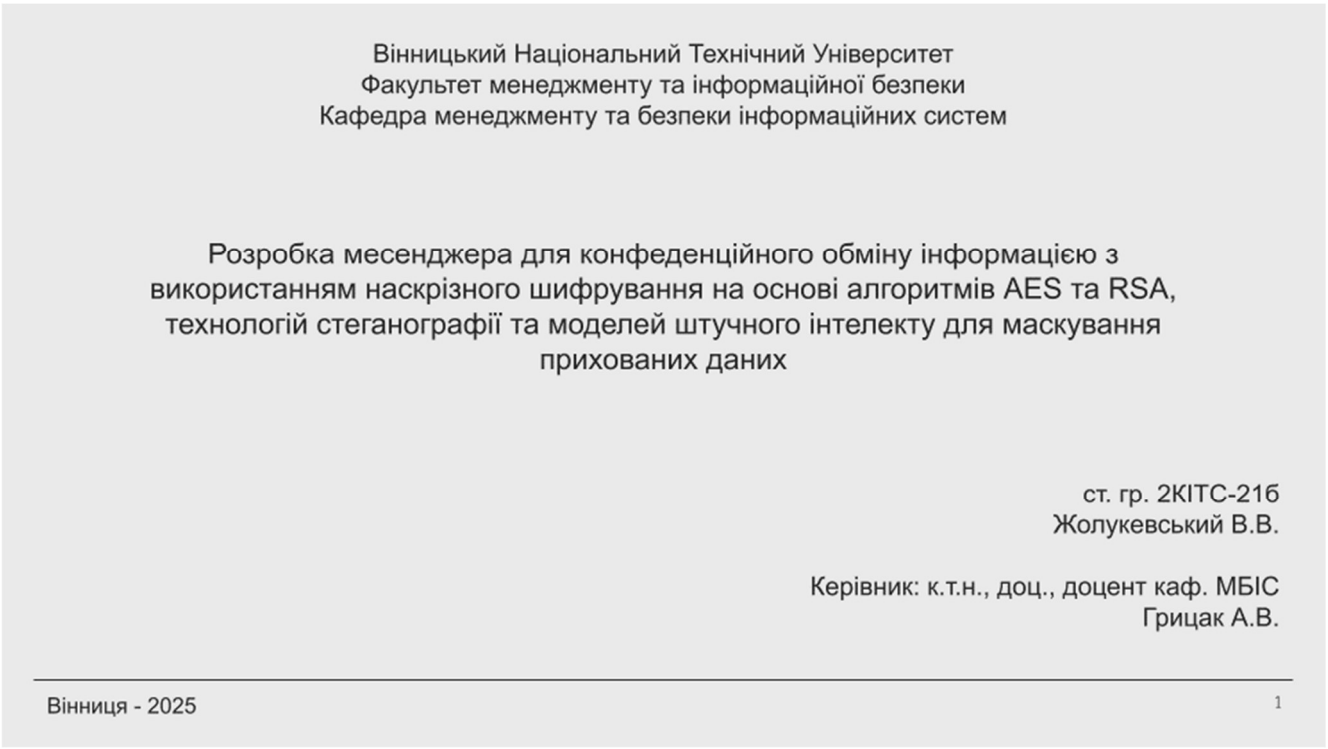

Рисунок 1 – початковий слайд презентації роботи

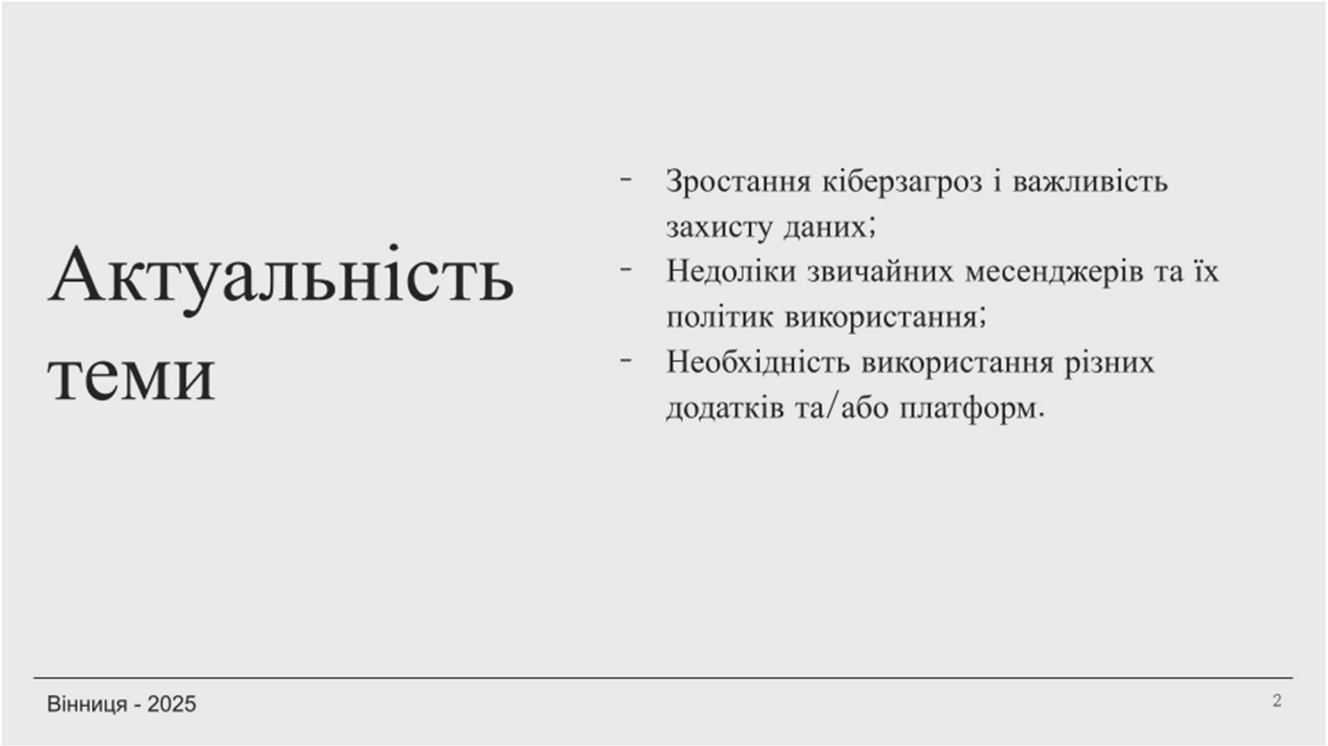

Рисунок 2 – слайд актуальності теми

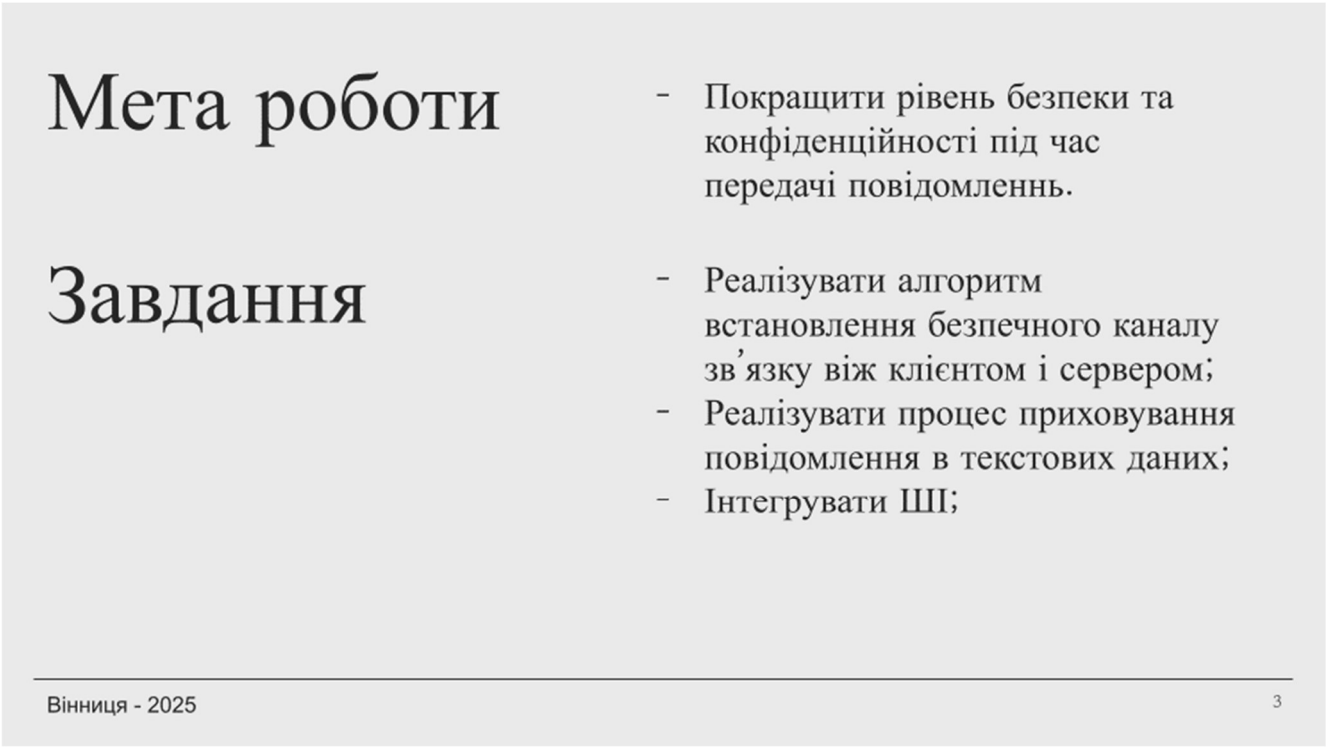

Рисунок 3 – слайд мети та завдання роботи

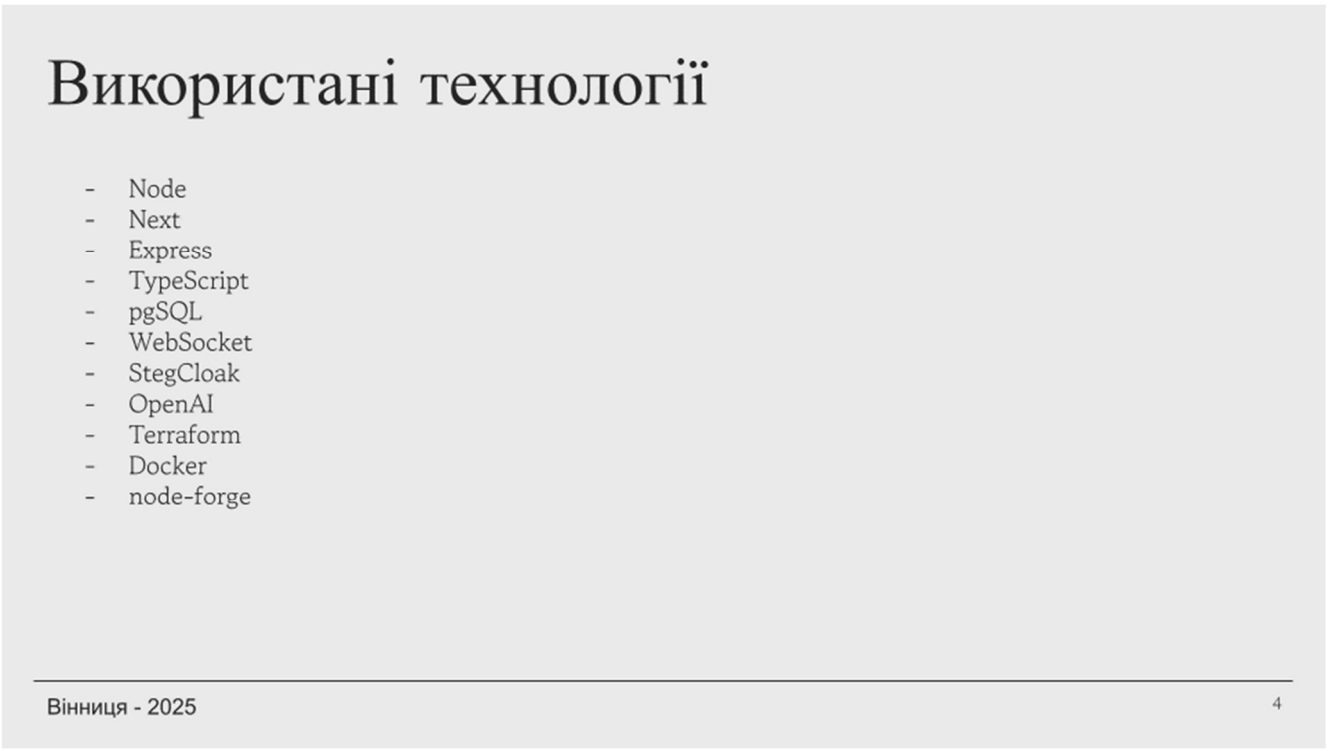

Рисунок 4 – слйди переліку використаних технологій

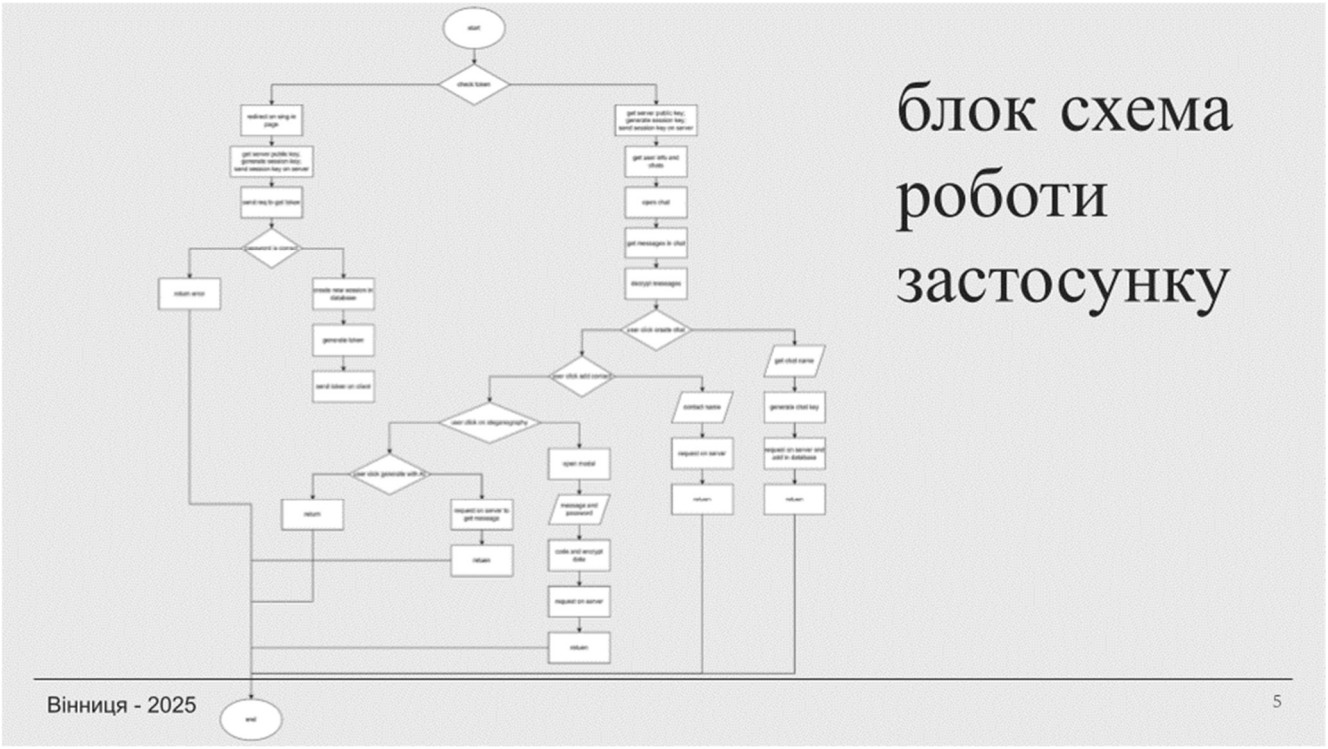

Рисунок 5 – слайд з зображенням блок схеми роботи додатку

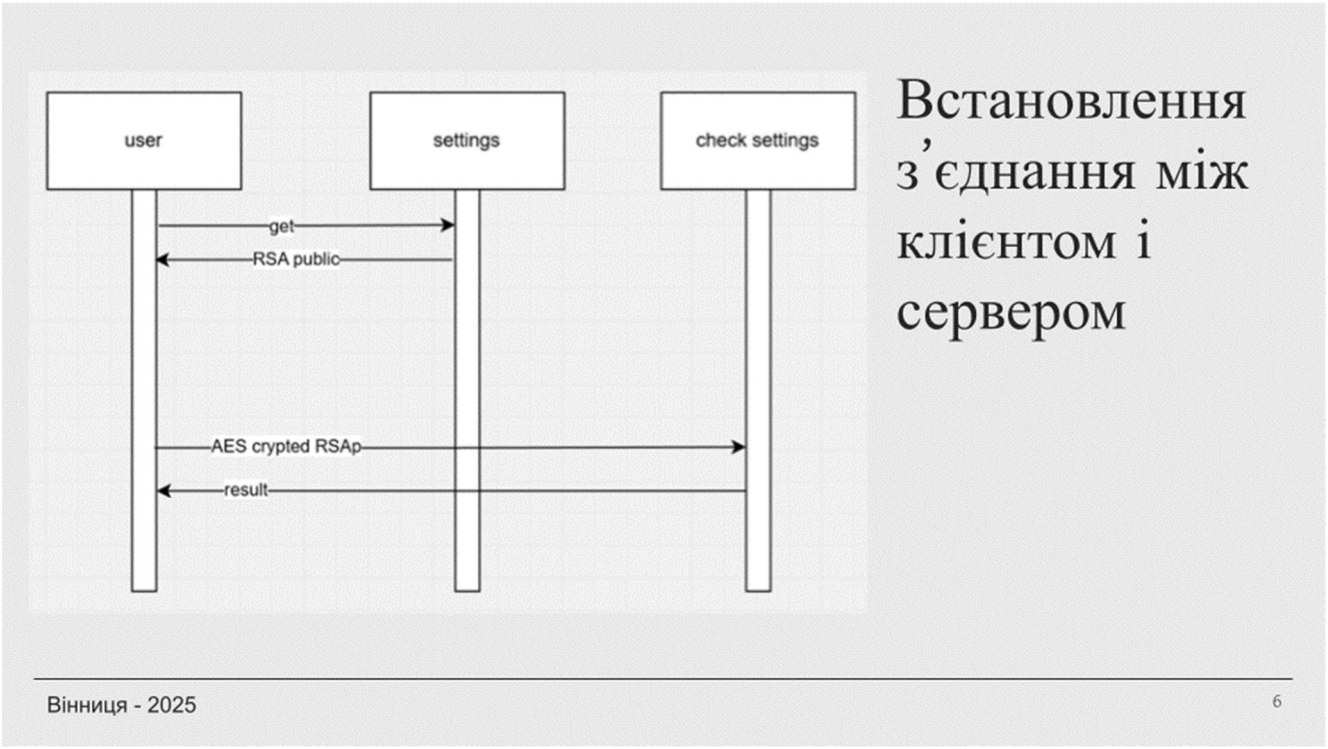

Рисунок 6 – слайд з зображенням діаграми послідовності для встановлення з’єднання між клієнтом і сервером

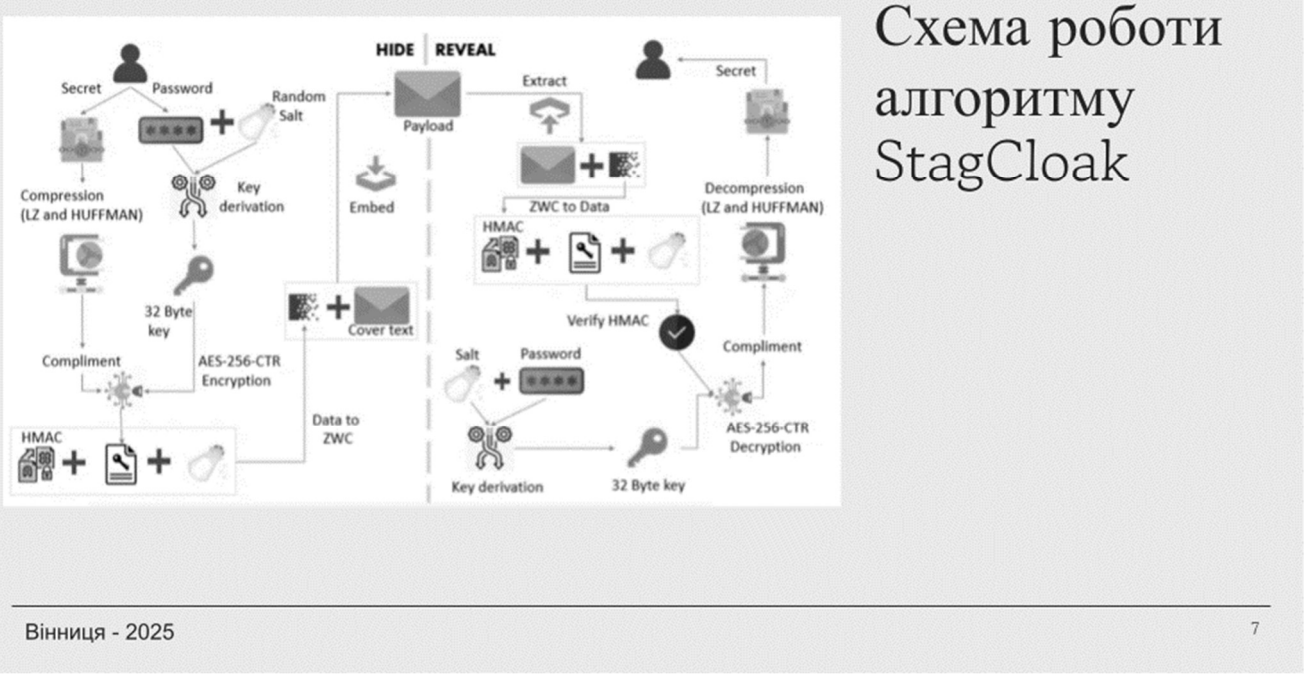

Рисунок 7 – слайд з зображенням алгоритму StagCloak

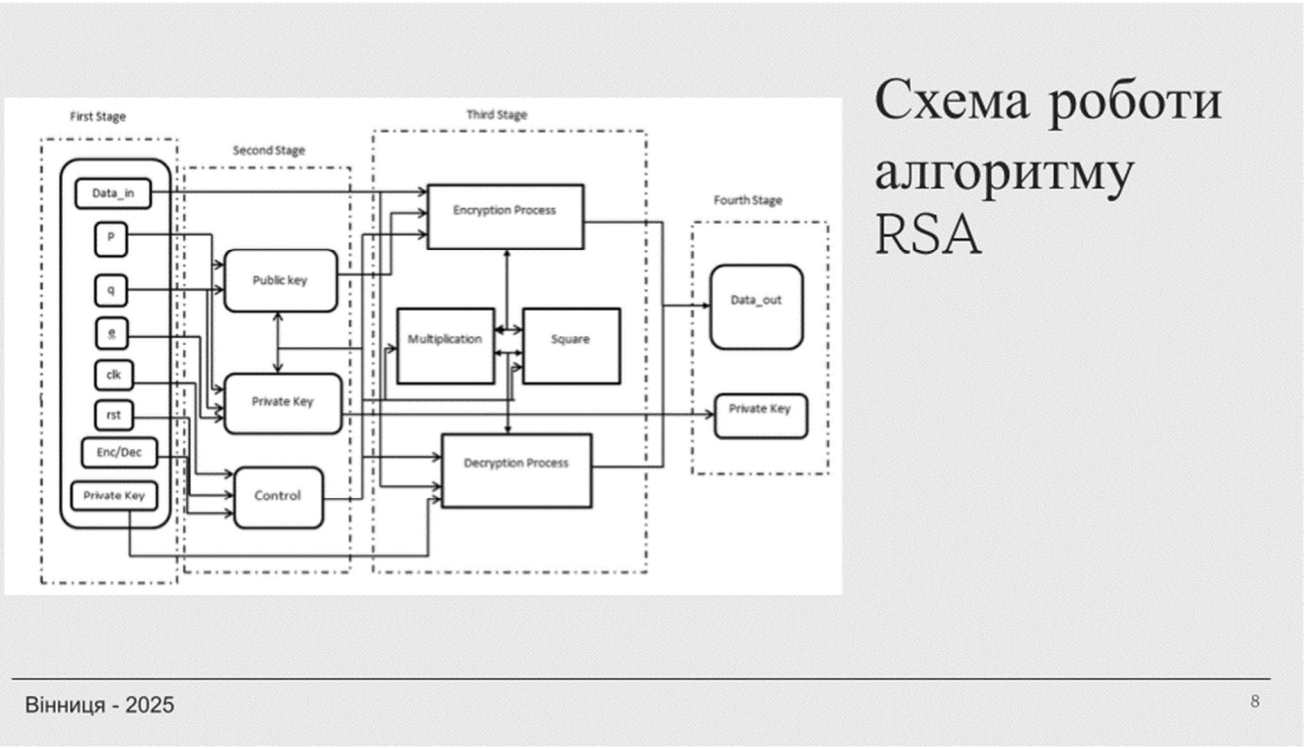

Рисунок 8 – слайд з зображенням алгоритму RSA

Рисунок 9 – слайд з зображенням алгоритму AES

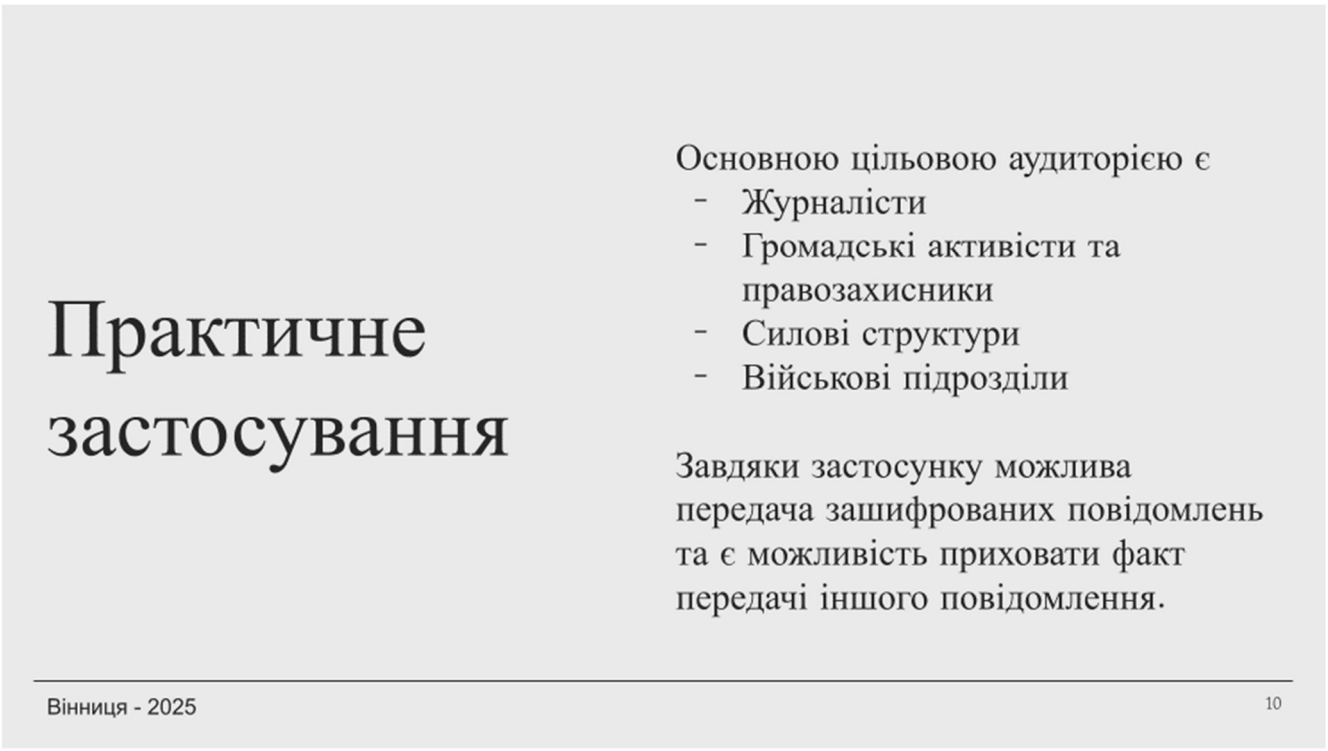

Рисунок 10 – слайд з переліком практичного застосування

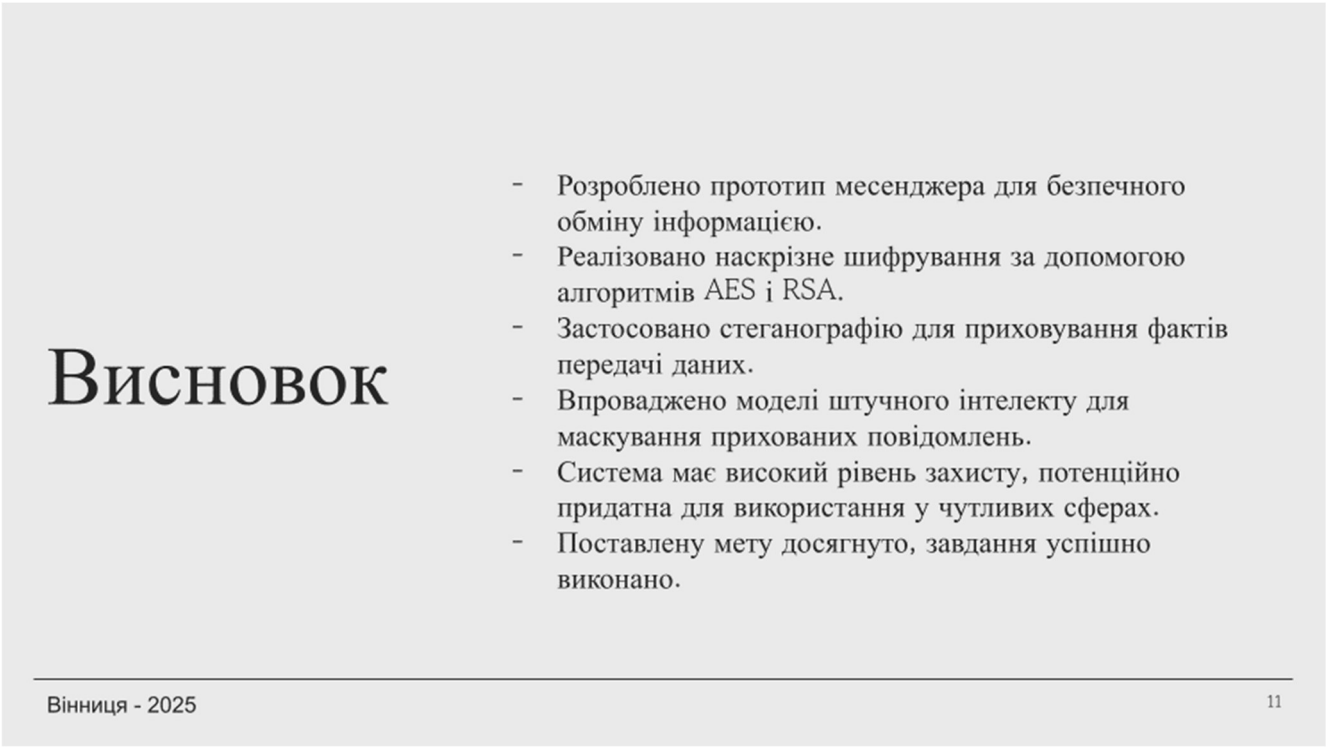

Рисунок 11 – слайд, висновок

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛПФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка месенджера для конфеденційного обміну інформацією з використанням наскрізного шифрування на основі алгоритмів AES та RSA, технологій стеганографії та моделей штучного інтелекту для маскування прихованих даних

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
факультет менеджменту та інформаційної безпеки
Гр. 2КІТС-21б

Керівник доцент Грицак А.В. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	98,95 %
Загальна схожість	1,05 %

Аналіз звіту подібності (відмітити потрібне)

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

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

Автор _____ Жолукевський В.В.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Допустити до захисту**

_____

_____

_____

_____

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

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