Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки та захисту
інформації
_____ Іван ПАРХОМЕНКО
« » червня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)
спеціальність 125 Кібербезпека
(код і назва спеціальності)
освітній ступень бакалавр
освітня програма Кібербезпека
(назва освітньо-професійної програми)
на тему: «Механізм шифрування трафіку для передачі даних через месенджери»

Виконавець: студент IV курсу, групи КБ-41

_____ **Павло СЛОБОДЯНИК**
(підпис) (ім'я, прізвище)

	Підпис	**Ім'я ПРІЗВИЩЕ**
Керівник		Юрій БАБЕНКО
Нормоконтроль		Інна МИХАЛЬЧУК

Київ 2025

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки та захисту інформації
_____Іван ПАРХОМЕНКО
«29» листопада 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студенту **КБ-41** **Слободянику Павлу Дмитровичу**
(група) (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Механізм шифрування трафіку для передачі даних через месенджери

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №6 від 28.11.2022 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Технічні аспекти передачі даних у месенджерах, використання наскрізного шифрування, принципи симетричного і асиметричного шифрування

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Огляд безпеки популярних месенджерів, загроз та механізмів захисту, огляд безпеки та конфіденційності даних у WhatsApp, відомих вразливостей безпеки, програмна реалізація механізму шифрування трафіку для передачі даних через месенджери

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Програмне рішення, що реалізує механізм шифрування трафіку для передачі даних через месенджери

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29 листопада 2024 року

Завдання видав _____ (підпис) Юрій БАБЕНКО (ім'я, прізвище)

Завдання прийняла до виконання _____ (підпис) Павло СЛОБОДЯНИК (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2024 – 05.12.2024	*виконано*
2	Аналіз літератури	05.12.2024 – 15.12.2024	*виконано*
3	Обґрунтування вибору рішення	15.12.2024 – 23.12.2024	*виконано*
4	Дослідження загроз безпеці та механізмів захисту у месенджерах	23.12.2024 – 12.01.2025	*виконано*
5	Огляд безпеки популярних месенджерів	12.01.2025 – 07.02.2025	*виконано*
6	Дослідження безпеки WhatsApp	07.02.2025 – 02.03.2025	*виконано*
7	Огляд відомих вразливостей та актуальних атак	02.03.2025 – 26.03.2025	*виконано*
8	Розробка програмного рішення шифрування трафіку для передачі даних через месенджери	26.03.2025 – 29.04.2025	*виконано*
9	Оформлення пояснювальної записки	29.04.2025 – 20.05.2025	*виконано*
10	Підготовка до захисту кваліфікаційної роботи	20.05.2025 – 13.06.2025	*виконано*

Завдання видав _____ (підпис) Юрій БАБЕНКО (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Павло СЛОБОДЯНИК (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 13 червня 2025 року

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, має 63 сторінки основного тексту, 1 таблицю та 26 рисунків. Список використаних джерел містить 30 найменувань і займає 4 сторінки.

Метою роботи є підвищення рівня безпеки передачі даних через месенджери, шляхом впровадження додаткового механізму шифрування.

Об'єктом дослідження є процес захищеного обміну інформацією через месенджери.

Предметом дослідження є механізм додаткового шифрування даних на стороні клієнта.

Для досягнення поставленої мети було визначено наступні завдання:

- Провести огляд популярних месенджерів, дослідження їх безпеки та збереження конфіденційності даних.
- Дослідити механізми безпеки WhatsApp, зокрема використання протоколу Signal для наскрізного шифрування.
- Проаналізувати атаки, які обходять механізми E2EE, зокрема фішинг та вразливості кінцевих пристроїв.
- Розробити програмний застосунок, що забезпечує механізм шифрування трафіку для передачі даних через месенджери.

*Практичною цінністю отриманих результат*ів є програмний застосунок, що забезпечує механізм шифрування трафіку для передачі даних через месенджери.

Ключові слова: месенджери, конфіденційність даних, наскрізне шифрування (E2EE), симетричне шифрування, AES.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕННЬ

AES	–	Advanced Encryption Standard
DH	–	Diffie-Hellman
E2EE	–	End-to-End Encryption
ECC	–	Elliptic Curve Cryptography
HMAC	–	Hash-based Message Authentication Code
MFA	–	Multi-Factor Authentication
MITM	–	Man-In-The-Middle
RSA	–	Rivest–Shamir–Adleman
TLS	–	Transport Layer Security
TOFU	–	Trust On First Use
UI	–	User Interface
X3DH	–	Extended Triple Diffie-Hellman
2FA	–	Two Factor Authentification

ВСТУП

У добу цифрової трансформації месенджери стали невід’ємною частиною комунікаційних процесів, охоплюючи особисте спілкування, ділову взаємодію та обмін конфіденційною інформацією. Станом на лютий 2025 року два мільярди користувачів щомісяця користувалися месенджером WhatsApp. Охоплення використання додатка особливо високе на ринках за межами Сполучених Штатів, і це один з найпопулярніших мобільних додатків у світі. Для Telegram і Facebook Messenger кількість користувачів у всьому світі – близько 950 мільйонів [1].

В Україні месенджери також набули широкого поширення: за дослідженням Kantar у 2022 серед користувачів Android 98% українців використовували Viber, 94% – Telegram 75% – Facebook Messenger , а 48% – WhatsApp [2].

Актуальність теми обумовлена тим, що зростання популярності месенджерів супроводжується підвищенням ризиків для безпеки даних, адже конфіденційна інформація, що передається через ці платформи, може стати мішенню для кіберзлочинців. Зокрема, в умовах війни в Україні месенджери відіграють критичну роль у передачі життєво важливої інформації, включаючи дані військового та особистого характеру. Таким чином, розробка програмного застосунку, що реалізує механізм шифрування трафіку для передачі даних через месенджери забезпечить підвищений рівень конфіденційності комунікацій.

Метою роботи є підвищення рівня безпеки передачі даних через месенджери, шляхом впровадження додаткового механізму шифрування.

Об'єктом дослідження є процес захищеного обміну інформацією через месенджери.

Предметом дослідження є механізм додаткового шифрування даних на стороні клієнта.

Для досягнення поставленої мети було визначено наступні завдання:

- Провести огляд популярних месенджерів, дослідження їх безпеки та збереження конфіденційності даних.
- Дослідити механізми безпеки WhatsApp, зокрема використання протоколу Signal для наскрізного шифрування.
- Проаналізувати атаки, які обходять механізми E2EE, зокрема фішинг та вразливості кінцевих пристроїв.
- Розробити програмний застосунок, що забезпечує механізм шифрування трафіку для передачі даних через месенджери.

*Практичною цінністю отриманих результат*ів є програмний застосунок, що забезпечує механізм шифрування трафіку для передачі даних через месенджери.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ЗАХИСТУ ДАННИХ У МЕСЕНДЖЕРАХ

1.1 Загрози безпеці при передачі даних у месенджерах

Месенджери є привабливою мішенню для кіберзлочинців через велику кількість конфіденційних даних, які через них передаються. Основними загрозами є:

- Атаки типу «людина посередині» (Man-in-the-Middle, MITM) виникають, коли зловмисник таємно втручається в комунікацію між двома сторонами, перехоплюючи або змінюючи повідомлення. Імітуючи легітимного учасника, атакуючий може обмінюватися криптографічними ключами з обома сторонами, отримуючи доступ до зашифрованих даних і пересилаючи повідомлення, залишаючись непоміченим. Такі атаки створюють ризики витоку конфіденційних даних, крадіжки особистих відомостей і компрометації інформації. Публічні Wi-Fi-мережі без шифрування або ненадійне SSL-pinning у клієнтських додатках можуть стати воротами для MITM-атак, коли трафік переадресовується через зловмисний проксі.

- Фішинг і соціальна інженерія є одними з найпоширеніших методів компрометації користувачів месенджерів. Фішингові атаки в месенджерах зазвичай включають надсилання підроблених повідомлень, які імітують офіційні сповіщення від платформи, наприклад, запити на підтвердження входу або пропозиції оновлення безпеки. Користувачів обманом змушують вводити облікові дані, коди двофакторної автентифікації (2FA) або переходити за шкідливими посиланнями, які встановлюють зловмисне ПЗ. Соціальна інженерія підсилює ці атаки, експлуатуючи довіру до контактів: зловмисники можуть використовувати скомпрометовані акаунти для розсилки фальшивих повідомлень від імені знайомих, пропонуючи файли чи посилання, які заражають пристрій жертви.

- Зловмисне програмне забезпечення (Malware). Шкідливі програми, такі як трояни чи шпигунське ПЗ, можуть бути доставлені через вкладення в месенджерах. Такі атаки дозволяють зловмисникам отримати доступ до пристрою, перехоплювати повідомлення або викрадати дані.

- Компрометація облікових записів. Зловмисники можуть отримати доступ до акаунтів через слабкі паролі, повторне використання паролів або вразливості в системах автентифікації. а також SIM-swap – заміну SIM-картки зловмисниками. Згідно з доповіддю Verizon Data Breach Investigations Report 2023, 74 % усіх інцидентів витоків даних спричинені людським фактором, насамперед ненадійними паролями та соціальною інженерією [3].

- Метаданні. Навіть при наскрізному шифруванні месенджери можуть збирати метадані (хто, коли і з ким спілкувався), які можуть бути використані для аналізу соціальних мереж або профілювання.

- Тиск урядів. Окрім прямих атак, значну загрозу несуть зусилля урядів із вимогами вбудовувати backdoors та обов’язково співпрацювати з правоохоронними органами. Наприклад у Великій Британії, до 2023 року неодноразово обговорювали поправки до Закону про розвідувальні повноваження (Investigatory Powers Act), що дозволяли б уряду примусово отримувати доступ до повідомлень у зашифрованих месенджерах для боротьби з тероризмом і кіберзлочинністю. Коли Apple відмовилася вбудувати такий backdoor у свій E2EE-режим Advanced Data Protection для iCloud, уряд змусив компанію вимкнути функцію E2EE для iCloud на території Велокої Британії [4]. Хоча подібні рішення дійсно важливі для збереження правопорядку, водночас вони несуть серйозні загрози конфіденційності, особливо в авторитарних режимах.

1.2 Механізми захисту в месенджерах

Передбачивши різноманітні загрози, сучасні месенджери імплементують низку механізмів захисту, кожен із яких вирішує певний набір завдань, але водночас має власні обмеження:

• Наскрізне шифрування (E2EE). Цей метод забезпечує конфіденційність повідомлень, роблячи їх доступними лише для відправника та отримувача. В E2EE дані шифруються на системі або пристрої відправника, і лише передбачуваний одержувач може їх розшифрувати. Під час передачі до місця призначення повідомлення не може бути прочитане або підроблене постачальником інтернет-послуг (ISP), зловмисником або будь-якою іншою організацією чи службою. Протокол Signal, який використовується в WhatsApp і Signal, є стандартом у цій сфері.

• Двофакторна автентифікація (2FA). Більшість месенджерів дозволяють налаштувати додатковий рівень захисту, наприклад, код із SMS або програми-аутентифікатора. Використання 2FA значно знижує ризик несанкціонованого входу в обліковий запис навіть у разі компрометації пароля. Згідно з даними, наведеними керівником національної безпеки США з кібербезпеки, використання багатофакторної автентифікації (MFA) може запобігти 80–90% кібератак [5].

• Відкритий вихідний код дозволяє стороннім експертам проводити аудит безпеки та виявляти вразливості до їхнього експлуатації. Наприклад Signal, має відкритий код, що забезпечує високий рівень довіри й прозорості. У той же час пропрієтарні месенджери такі як Viber та WhatsApp, не публікують ані серверну, ані клієнстьку частину коду, через що користувачі не можуть пересвідчитися в повноті реалізації E2EE й відсутності бекдорів.

Незважаючи на ці механізми, жоден месенджер не є абсолютно безпечним. Наприклад, наскрізне шифрування не захищає від фізичного доступу до пристрою чи атак соціальної інженерії. Крім того, завжди залишаються вразливості в самій архітектурі програмного забезпечення.

1.3 Огляд популярних месенджерів та їх особливостей

Сьогодні ринок месенджерів надзвичайно різноманітний, він пропонує десятки додатків, кожен із яких вирізняється унікальними особливостями,

цільовою аудиторією та підходами до забезпечення безпеки. Ці платформи задовольняють широкий спектр потреб користувачів – від тих, хто цінує максимальну конфіденційність, до тих, хто шукає багатофункціональні екосистеми з інтеграцією різноманітних сервісів. Для подальшої роботи необхідно проаналізувати найпопулярніші месенджери, дослідити їхні функціональні можливості, особливості дизайну та стратегії захисту даних, щоб отримати глибше розуміння їхніх сильних і слабких сторін у контексті сучасних вимог до безпеки та зручності.

1.3.1 WhatsApp

WhatsApp – провідний месенджер у світі, який має понад 2 мільярди користувачів у 180 країнах. Він використовує наскрізне шифрування (E2EE) на основі протоколу Signal, що гарантує, що повідомлення, дзвінки та медіа доступні лише відправнику та одержувачу. E2EE WhatsApp застосовується до всіх чатів, включаючи групи, і воно ввімкнено за замовчуванням [6]. Також є можливість використання E2EE для хмарних бекапів. Це означає, що повідомлення шифруються на пристрої відправника і розшифровуються лише на пристрої отримувача, а компанія-власник (Meta) не має доступу до вмісту переписки. Проте WhatsApp критикують за збір метаданих (наприклад, номери телефонів, час відправлення повідомлень), які можуть використовуватися для профілювання користувачів.

У 2021 році GDPR (General Data Protection Regulation) оштрафувала WhatsApp на 225 мільйонів євро, за порушення регуляцій [7]. Загальний регламент про захист даних (GDPR) – це один з найжорсткіших у світі законів про конфіденційність та безпеку. Хоча його було розроблено та прийнято Європейським Союзом (ЄС), він накладає зобов’язання на організації будь-де, якщо вони збирають дані, пов’язані з громадянами ЄС. Було встановлено, що месенджер забезпечує недостатню прозорість щодо того, як він обробляє дані користувачів і навіть некористувацькі дані. Наприклад, WhatsApp запитує доступ

до всього списку телефонних контактів користувача та даних, що зберігаються в них. Хоча заявленою метою WhatsApp є пошук контактів, що належать користувачам одного й того ж месенджера, та допомога у швидкому та легкому зв'язку з ними, насправді він отримує доступ навіть до контактів, які не є його користувачами. Таким чином, месенджер порушує правило, яке вимагає явної згоди на збір та обробку інформації. [8]

Більше того, WhatsApp також дозволив обмін контактами своїх користувачів з материнською компанією Facebook. Його політика конфіденційності не була зовсім прозорою щодо цього та не надавала чіткого опису того, які дані передаються та чому. Під тиском соціальних мереж та ЗМІ WhatsApp оновив свою політику захисту даних користувачів, але це не врятувало месенджер від штрафу за порушення GDPR.

1.3.2 Signal

Signal – це месенджер з відкритим кодом, що є еталоном приватності та безпеки. Розроблений некомерційною організацією Signal Foundation, він має відкритий вихідний код, не містить реклами та не збирає особисту інформацію користувачів. Signal був створений криптографом та підприємцем Моксі Марлінспайком який також обіймав посаду генерального директора компанії з 2015 по 2022 рік [9].

Станом на 2025 рік Signal використовує близько 40-70 мільйонів користувачів щомісяця, що є зовсім не великими цифрами порівняно з найбільшими месенджерами такими як WhatsApp та Messenger, що налічують мільярди клієнтів [10].

Signal використовує однойменний Signal Protocol, що також ліг в основу наскрізного шифрування, який складається з:

- X3DH для первинного обміну ключами.
- Double Ratchet для генерування ключів із forward secrecy.
- AES-256 для симетричного шифрування.

• HMAC-SHA256 для цілісності повідомлень.

Коли ви користуєтеся Signal, ваші дані зберігаються в зашифрованому вигляді на ваших пристроях. Єдина інформація, яка зберігається на серверах Signal для кожного облікового запису це номер телефону, за яким ви зареєструвалися, дата та час реєстрації в сервісі, а також дата останнього входу в систему.

Як зазначає Signal, застосунок не зберігає інформацію про контакти користувача (наприклад, самі контакти, хеш контактів, будь-яку іншу похідну контактну інформацію), інформацію про групи користувача (наприклад, у скількох групах перебуває користувач, у яких групах перебуває користувач, списки учасників груп користувача) або будь-які записи про те, з ким спілкувався користувач [11].

1.3.3 Telegram

Telegram – кросплатформений месенджер, що має функціонал наскрізно зашифрованих чатів (більш відомі, як «секретні чати»), відеодзвінки VoIP, обмін файлами та деякі інші функції.

Telegram застосовує власний протокол MTProto 2.0, який для cloud-чатів використовує гібрид SSL-подібного каналу та кастомних крипто алгоритмів, але лише “секретні чати” отримують справжнє E2EE між пристроями. Відкритість вихідного коду у протоколі MTProto дозволяє експертам з безпеки переглядати код та виявляти потенційні вразливості. Однак, на відміну від Signal, який повністю має відкритий вихідний код, серверний код Telegram не є загальнодоступним, а це означає, що деякі аспекти його роботи залишаються недоступними для дослідження.

У підсумку, до безпеки та конфіденційності Telegram залишаються питання:

• Метадані та інформація про місцезнаходження. Хоча повідомлення можуть бути зашифрованими, метадані, такі як хто, коли та де надіслав

повідомлення, все ще можуть бути доступними, що може розкрити конфіденційну інформацію про схеми зв'язку.

- Ризики, пов'язані з хмарним сховищем. Звичайні чати зберігаються в хмарі та не шифруються наскрізно, а це означає, що ці повідомлення доступні для Telegram, хоча й у зашифрованому вигляді.

- Вразливості та загрози безпеці. Дослідники кібербезпеки вказали на потенційні недоліки протоколу MTProto Telegram, припускаючи, що він може бути не таким безпечним, як інші протоколи обміну повідомленнями, подібні до тих, що використовуються Signal [12].

- Співпраця з урядами. Практика шифрування Telegram призвела до конфліктів з урядами, особливо в країнах, які вимагають доступу до даних користувачів. За словами українських спецслужб, Telegram співпрацює з російською владою та ФСБ, блокуючи канали на їхнє прохання та зберігаючи дані користувачів необмежений час [13].

1.3.4 Viber

Viber – це кросплатформений месенджер, який з 2014 року належить японській компанії Rakuten. Наразі він має понад 1 мільярд користувачів по всьому світу, значна частина аудиторії яких з Європи та Близького Сходу.

З 2016 року Viber впровадив E2EE за замовчуванням для всіх індивідуальних і групових чатів, дзвінків та обміну медіафайлами. Протокол шифрування Viber оснований на протоколі Open Whisper Systems (Signal Protocol), але модифікований під архітектуру Viber [14]. Кожне повідомлення шифрується окремим ключем (принцип попередньої безпеки) і ключі зберігаються тільки на пристроях користувачів.

Додаток реалізує функцію «довірених контактів», яка аутентифікує співрозмовника шляхом обміну секретними ключами (QR-кодами або ручним введенням). Viber також має приховану кореспонденцію, яка дозволяє створювати окремі чати, захищені паролем, без історії повідомлень.

Політику конфіденційності Viber критикували дослідники конфіденційності, оскільки вона дозволяє ділитися даними користувачів для профілювання та покращення рекламних послуг, які надає Rakuten. Viber зберігає деякі метадані, такі як список контактів, часові мітки повідомлень і інформацію про пристрій [15].

1.3.5 Facebook Messenger

Facebook Messenger – один з найпопулярніших месенджерів у світі, і станом на 2024 рік має приблизно 980 мільйонів активних користувачів щомісяця [1]. Розроблений компанією Meta Platforms Inc. (раніше відомою як Facebook Inc).

Подібно до Telegram, Facebook Messenger не використовує наскрізне шифрування за замовчуванням, а працює за клієнт-серверною моделлю, де клієнтські пристрої (смартфони, комп'ютери) взаємодіють із централізованими серверами Meta через протокол HTTPS із застосуванням Transport Layer Security для захисту даних під час передачі. Повідомлення шифруються на пристрої відправника, передаються на сервери Meta, які зберігають метадані (наприклад, контакти, час надсилання, IP-адреси), і доставляються одержувачу.

У 2022 році Meta Platforms Inc. розпочала тестове впровадження наскрізного шифрування (E2EE) за замовчуванням для звичайних чатів у Facebook Messenger, обмеживши експеримент окремими групами користувачів. Цей крок став відповіддю на тривалу критику щодо відсутності E2EE за замовчування, на відміну від WhatsApp і Signal. Проте реалізація була поступовою, і, за словами представників компанії, повне охоплення E2EE може тривати роками, оскільки технологія складна й потребує глибокої інтеграції в архітектуру платформи [16] .

Facebook Messenger є однією з найбільш критикованих платформ через агресивну політику збору даних. Згідно з політикою конфіденційності Meta, Messenger отримує доступ до широкого спектру даних, включаючи список контактів, IP-адреси, технічні характеристики пристрою, історію дій у додатку,

час і тривалість дзвінків, геолокацію, профільну інформацію з Facebook (ім'я, аватар, зв'язки), взаємодію з бізнесами та навіть аудіозаписи при використанні голосового вводу. Ці дані використовуються для персоналізації реклами, аналітики та інтеграції з іншими сервісами Meta, такими як Instagram і Facebook.

1.3.6 Результат порівняння

Таблиця 1.1

Порівняння досліджуваних месенджерів

Месенджер	**E2EE за замовчування**	**Відкритий код**	**Збір метаданих**	**Хмарні резервні копії з E2EE**
WhatsApp	так	ні	так	Google Drive або iCloud з E2EE
Signal	так	так	ні	Бекапи зберігаються локально
Telegram	«секретні чати»	відкритий клієнт	так	Всі дані зберігаються на серверах Telegram
Viber	так	ні	так	Google Drive або iCloud без E2EE
Messenger	«таємні чати»	ні	так	Всі данні зберігаютсья на серверах Мета

Результати порівняння наведені в табл. 1.1. Серед порівнюваних месенджерів, Signal і WhatsApp є двома платформи з вищим рівнем захисту порівняно з іншими. Signal - це єдина платформа з повністю відкритим вихідним кодом, не зберігає жодних метаданих, крім номера телефону та часів входу, і уникає хмарних резервних копій, що фактично виключає доступ сторонніх до

даних. Хоча WhatsApp дійсно збирає метадані, водночас месенджер забезпечує E2EE за замовчуванням для всіх чатів і опціональне шифрування резервних копій, що зберігаються в хмарі, роблячи його насамперед доступним і одночасно відносно безпечним рішенням. Інші месенджер демонструють менший рівень безпеки, так як або не мають E2EE за замовчуванням (Telegram, Facebook Messenger), або мають закритий код з певними ризиками збору метаданих (Viber), що знижує їхню загальну безпеку.

Висновки за розділом 1

У розіділі 1 було розглянуто актуальність проблеми захисту конфіденційності в месенджерах, яка набуває особливого значення в умовах глобальної цифровізації. В Україні месенджери відіграють критично важливу функцію не лише в особистих і професійних комунікаціях, але й у контексті обміну життєво важливою інформацією в умовах воєнного часу.

Основними загрозами безпеці є фішинг та соціальна інженерія, що потенційно загрожує компрометацією акаунту або зараженням шкідливим шпигунським ПЗ. У разі не використання наскрізного шифрування можлива реалізація таких атак, як MITM, що призводять до перехоплення або підміни повідомлень. Непрозорі політики безпеки компаній, особливо щодо збору та використання метаданих є причинами регулярних судових позовів від регуляторів.

Серед механізмів захисту найважливішим є наскрізне шифруванн E2EE, що забезпечує конфіденційність повідомлень, роблячи їх доступними лише для відправника та отримувача та двофакторна аутентифікація, наявність якої значно зменшує ризики компрометації акаунту. Відкритий вихідний код, як в Signal дозволяє незалежний аудит та покращує рівень безпеки.

Серед проаналізованих месенджерів найвищий рівень безпеки демонструє Signal, завдяки повністю відкритому вихідному коду, реалізації криптографічного протоколу Signal Protocol і політиці мінімального збору

метаданих. Водночас WhatsApp, який використовує той самий протокол шифрування, залишається частиною екосистеми Meta, що викликає занепокоєння щодо збору інформації про користувачів. Telegram, Viber і Facebook Messenger пропонують E2EE лише частково або опціонально.

РОЗДІЛ 2
АНАЛІЗ ЗАХИЩЕНОСТІ WHATSAPP

2.1 End-To-End Encryption

Наскрізне шифрування (E2EE) – це безпечний механізм передачі даних, який гарантує, що лише користувачі, що беруть участь у спілкуванні мають доступ до повідомлень. Наскрізне шифрування (E2EE) вважається найприватнішим та найбезпечнішим способом зв'язку.

Подібно до інших методів шифрування, E2EE перетворює читабельний відкритий текст на нечитабельний зашифрований текст за допомогою криптографії. Цей процес допомагає маскувати конфіденційну інформацію від неавторизованих користувачів і гарантує, що лише цільові одержувачі з правильним ключем розшифрування можуть отримати доступ до конфіденційних даних.

Однак E2EE відрізняється від інших методів шифрування, оскільки забезпечує безпеку даних від початку до кінця. Він шифрує дані на пристрої відправника, зберігає їх зашифрованими під час передачі та розшифровує їх лише тоді, коли вони досягають кінцевої точки одержувача. Цей процес гарантує, що постачальники послуг, сервіси, що забезпечують обмін повідомленнями, такі як WhatsApp, не можуть отримати доступ до повідомлень. Тільки відправник і цільовий одержувач можуть їх прочитати.

Для порівняння, шифрування під час передачі захищає дані лише під час їх переміщення між кінцевими точками. Наприклад, протокол шифрування Transport Layer Security (TLS) шифрує дані під час їх передачі між клієнтом і сервером. Однак він не забезпечує надійного захисту від доступу посередників, таких як сервери додатків або мережеві провайдери.

Стандартне шифрування під час передачі часто є ефективнішим, але багато людей та організацій побоюються ризику доступу постачальників послуг до їхніх

конфіденційних даних. Будь-яке викриття, навіть на рівні кінцевої точки, може серйозно загрожувати конфіденційності даних та загальній кібербезпеці. Тому E2EE вважається золотим стандартом захисту конфіденційних даних у цифрових комунікаціях.

E2EE включає такі чотири кроки (рис. 2.1)

Рисунок 2.1 – Принцип End-To-End-Encryption [17]

1) Шифрування

E2EE (наскрізне шифрування) починається з використання криптографічного алгоритму для шифрування конфіденційних даних. Такий алгоритм застосовує складні математичні операції для перетворення повідомлень у нечитабельний формат, відомий як шифротекст. Розшифрувати ці дані можуть лише авторизовані користувачі, які мають відповідний секретний ключ.

E2EE поєднує два підходи до шифрування: симетричне (де один спільний ключ використовується як для шифрування, так і для дешифрування) та асиметричне (де використовуються пара ключів – публічний і приватний). Асиметричне шифрування застосовується для безпечного обміну симетричними ключами, які, своєю чергою, використовуються для захищеного передавання самих повідомлень.

2) Передача

Зашифровані дані передаються каналом зв'язку, таким як Інтернет або інші мережі. Повідомлення залишається нечитабельним для серверів додатків, постачальників інтернет-послуг (ISP), хакерів або інших організацій, коли воно рухається до місця призначення. Натомість воно виглядає як випадкові, незрозумілі символи для будь-кого, хто може його перехопити.

3) Розшифрування

Коли зашифроване повідомлення досягає пристрою одержувача, воно розшифровується за допомогою симетричного ключа, який раніше був захищено переданий з використанням асиметричного шифрування. Приватний ключ одержувача застосовується лише на етапі обміну ключами, а самі повідомлення дешифруються симетричним ключем. Таким чином, тільки одержувач, який має відповідну пару ключів, здатен відновити зміст повідомлення.

4) Автентифікація

Розшифровані дані перевіряються для забезпечення їх цілісності та автентичності. Цей крок може включати перевірку цифрового підпису відправника або інших облікових даних, щоб підтвердити, що ніхто не підробляв дані під час передачі.

Наскрізне шифрування унікальне в порівнянні з іншими системами шифрування тим, що використовує переваги і симетричної і асиметричної криптографії (рис. 2.2).

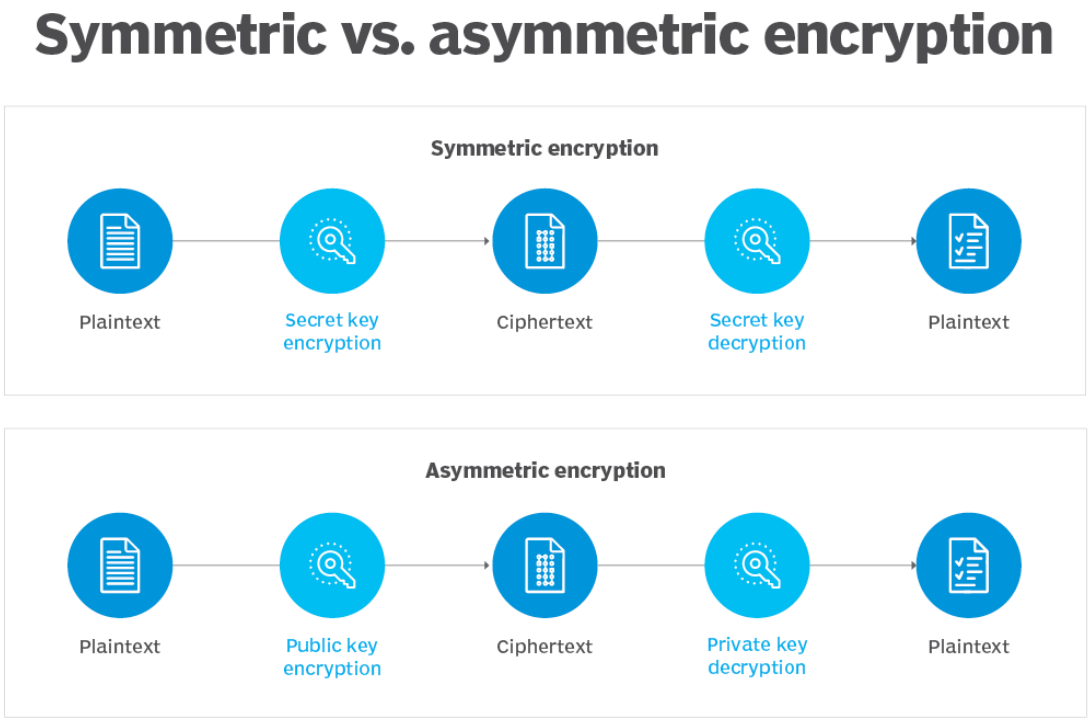

Рисунок 2.2 – Симетричне та асиметричне шифрування [19]

Симетричне шифрування – це метод, при якому один і той самий ключ використовується як для шифрування, так і для дешифрування повідомлень. Цей ключ може бути паролем, кодовою фразою або випадково згенерованим набором байтів.

Перевагами симетричного шифрування є висока продуктивність, алгоритми симетричного шифрування працюють значно швидше ніж асиметричні алгоритми. Недоліками є проблема обміну ключем, оскільки обидві сторони мають знати секретний ключ, має існувати захищений канал передачі ключа.

Асиметричне шифрування – це метод, при якому використовуються дві пари ключів: публічний ключ (public key) для шифрування і приватний ключ (private key) для дешифрування. Публічний ключ можна відкрито передавати іншим, тоді як приватний зберігається в секреті.

Однією з головних переваг є те, що воно усуває необхідність обміну секретними ключами, що може бути складним процесом, особливо під час спілкування з кількома сторонами. Крім того, асиметричне шифрування дозволяє створювати цифрові підписи, які можна використовувати для перевірки автентичності даних. Прикладами алгоритмів асиметричного шифрування є RSA, Діффі-Хеллмана та криптографія еліптичних кривих (ECC) [18].

Коли два користувачі починають розмову в WhatsApp, вони генерують унікальний сеансовий ключ для цієї конкретної розмови. Цей сеансовий ключ дозволяє симетрично шифрувати та дешифрувати повідомлення, якими обмінюються під час розмови. Сеансовий ключ передається через асиметричну систему шифрування. Він шифрується відкритим ключем одержувача та дешифрується його закритим ключем, що означає, що зловмисники не можуть вкрасти його під час передачі.

Цей комбінований метод дозволяє користувачам скористатися як безпекою асиметричного шифрування, так і ефективністю симетричного шифрування.

2.2 Signal Protocol

Протокол Signal – це криптографічний протокол, який забезпечує наскрізне шифрування (E2EE) для текстових повідомлень, голосових і відеодзвінків у месенджерах, таких як Signal, WhatsApp і Facebook Messenger (у режимі Secret Conversations). Розроблений Open Whisper Systems у 2013 році, він поєднує кілька криптографічних механізмів для забезпечення конфіденційності, цілісності, автентифікації та стійкості до атак. [20]

Протокол Signal складається з двох основних компонентів: Extended Triple Diffie-Hellman (X3DH) для ініціації сеансу та Double Ratchet для шифрування повідомлень під час комунікації. Вони використовують криптографічні примітиви Curve25519, AES-256 і HMAC-SHA256, забезпечуючи асинхронність, стійкість до компрометації та захист від атак. На рис. 2.3 зображена схема роботи протоколу Signal у WhatsApp.

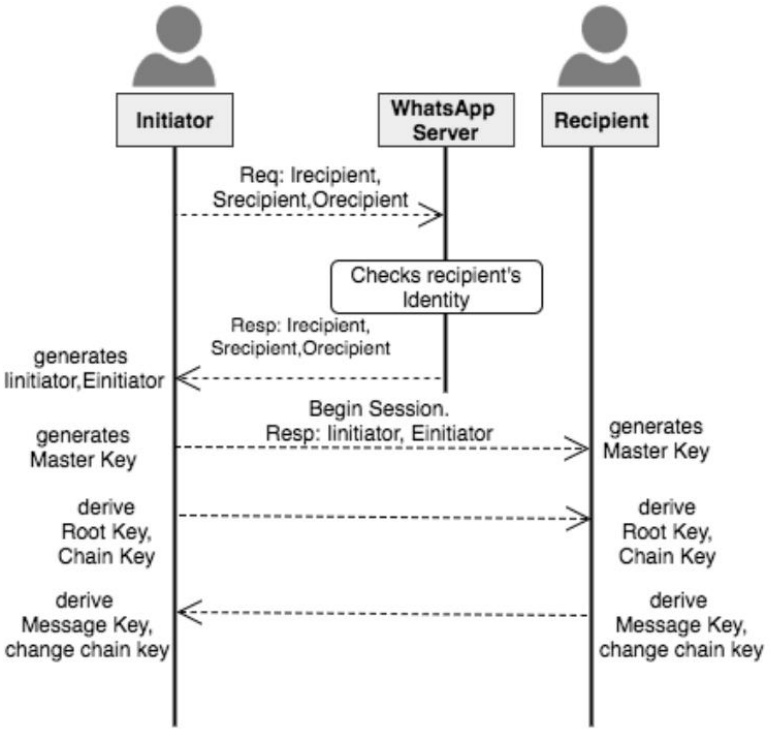

Рисунок 2.3 – Signal Protocol у WhatsApp [22]

2.2.1 Ініціація сеансу (X3DH)

X3DH (Extended Triple Diffie-Hellman) – це протокол обміну ключами, який встановлює спільний секрет між двома сторонами (наприклад, Алісою і Бобом) для подальшого шифрування. Створює спільний секретний ключ для ініціації E2EE, навіть якщо одна сторона офлайн [21].

1) Кожен користувач генерує такі ключі

- Identity Key (IK): довгострокова пара ключів (IK_private, IK_public).
- Signed PreKey (SPK): середньострокова пара ключів (SPK_private, SPK_public), підписана за допомогою IK_private.
- One-Time PreKeys (OPK): короткострокові ключі, що завантажуються на сервер і використовуються одноразово.

2) Запит Аліси

- Аліса отримує «пакет попередніх ключів» (prekey bundle) Боба з сервера, який включає його Identity Key (IK), підписаний попередній ключ (SPK) і одноразовий ключ (якщо доступний).

3) Обчислення Elliptic curve Diffie–Hellman

Аліса генерує Ephemeral key (EK) – ефемерний ключ і виконує три або чотири обміни Diffie-Hellman (DH):

- DH1 = DH(IK_Alice_private, SPK_Bob_public) між ідентифікаційним ключем Аліси та підписаним попереднім ключем Боба.
- DH2 = DH(EK_Alice_private, IK_Bob_public) між ефемерним ключем Аліси та ідентифікаційним ключем Боба.
- DH3 = DH(EK_Alice, SPK_Bob) між ефемерним ключем Аліси і Signed PreKey (SPK) Боба.
- DH4 = DH(EK_Alice, OPK_Bob) між ефемерним ключем Аліси та одноразовим попереднім ключем Боба (за його наявності). Ці обміни комбінуються для створення Secret_Key (SK).

4) Обчислення Secret Key

- SK = KDF(DH1 || DH2 || DH3 || DH4), результати обчислень обробляються через Key Deriviation Function (KDF), щоб отримати Secret Key, з якого за допомогою KDF будуть згнеровані Root_Key і Chain_Key для Double Ratchet.

Асинхронність дозволяє Алісі надсилати повідомлення, навіть якщо Боб офлайн, використовуючи попередні ключі. Процес автентифікації і підписування ключів запобігають атакам «людина посередині» (MITM).

2.2.2 Шифрування повідомлень (Double Ratchet)

Double Ratchet – це механізм, який забезпечує шифрування повідомлень після ініціації сеансу, поєднуючи DH Ratchet і Symmetric Ratchet для динамічного оновлення ключів. Таким чином забезпечується forward secrecy і post-compromise security, оскільки генеруютсья унікальні ключі для кожного повідомлення.

1) Кореневий ланцюг (Root Chain)

- Використовує Root_Key, отриманий із X3DH.
- Коли одна зі сторін отримує новий DH публічний ключ від партнера, вона виконує новий DH обмін: DH(DHs_private, DHr_public) Передає цей результат до KDF разом із поточним Root Key
- Таким чином KDF генерує новий RK і CK: RK', CK = KDF_RK(RK, DH_output)
- Це забезпечує post-compromise security, оскільки компрометація одного ключа не впливає на майбутні сеанси.

2) Ланцюг відправлення/отримання (Sending/Receiving Chain)

- Використовує Chain_Key, отриманий із Root_Key
- Для кожного нового повідомлення використовується Symetrical-key Ratchet для генерації унікального Message Key, який обчислюється за допомогою KDF: message_key, new_chain_key = KDF_CK(current_chain_key)

- Кожне повідомлення шифрується за допомогою унікального Message Key застосовуючи шифрування AES-256 у режимі CBC або CTR + HMAC-SHA256.

3) Надсилання і отримання повідомлення

- Аліса шифрує повідомлення, використовуючи Message Key, і оновлює Chain Key для наступного повідомлення.

- Боб отримує повідомлення, визначає його індекс і генерує відповідний Message Key для розшифрування повідомлення.

4) Оновлення ключів

- DH Ratchet активується при отриманні нового DH публічного ключа, оновлюючи кореневий ключ, з якого будуть генеруватися sending Chain Key для надсилання і receiving Chain Key для розшифрування повідомлень відвовідно.

- Symmetric Ratchet оновлюється для кожного повідомлення, забезпечуючи forward secrecy.

2.2.3 Групові чати

Signal Protocol підтримує E2EE для групових чатів, поєднуючи попарний Double Ratchet із мультикастовим шифруванням:

- При створенні групи або надсиланні першого повідомлення, відправник генерує Sender Key — симетричний ключ, який надалі використовується для шифрування повідомлень у цій групі.

- Sender Key шифрується індивідуально через Double Ratchet для кожного члена групи і надсилається окремо через наявні парні канали (засновані на X3DH + Double Ratchet), що забезпесує forward secrecy на етапі обміну ключами.

- Далі відправник застосовує симетричний механізм оновлення (Symmetric-key Ratchet) до Sender Key, генеруючи нові ключі для кожного повідомлення

- При зміні складу групи або компрометації генерується новий Sender Key, який знову поширюється через парні канали.

Таким чином, Signal Protocol забезпечує:

- Forward secrecy: компрометація поточного ключа не дозволяє розшифрувати попередні повідомлення.
- Post-compromise security: нові ключі обмежують вплив компрометації.
- Стійкість до втрати порядку: протокол обробляє повідомлення, отримані не в порядку надсилання.
- Автентифікацію: Користувачі можуть порівнювати відбитки відкритих ключів (fingerprint) через зовнішній канал для перевірки ідентичності, запобігаючи MITM-атакам.
- Довіру при першому використанні (TOFU): система сповіщає про зміну ключів контакту, сигналізуючи про можливу атаку.
- Заперечення (deniability): X3DH забезпечує криптографічну можливість заперечення авторства повідомлень
- Асинхронність: підтримка офлайн-повідомлень через попередні ключі.

2.3 Безпека та конфіденційність даних у WhatsApp

Використання наскрізного шифрування (E2EE), зокрема протоколу Signal, забезпечує високий рівень безпеки комунікації. Завдяки механізму безперервного оновлення криптографічних ключів (ratcheting) та обміну Diffie-Hellman, протокол значно ускладнює можливість проведення атак типу «людина посередині» (MITM) та гарантує властивість forward secrecy. Однак, незважаючи на ці сильні криптографічні гарантії, залишаються вразливості, які не покриваються або не залежать від самої реалізації E2EE. До них належать проблеми, пов’язані з особливостями програмної реалізації додатку, а також соціальною інженерією. Таким чином, конфіденційність не може бути гарантована виключно застосуванням E2EE і потребує комплексного підходу.

2.3.1 Безпека кінцевих пристроїв і хмарного зберігання

Незважаючи на надійність E2EE у процесі передачі даних, значною проблемою залишається безпека кінцевих пристроїв, таких як смартфони, планшети чи комп'ютери. Більшість Android-пристроїв не забезпечують наскрізне шифрування локальних даних. WhatsApp з певною періодичністю автоматично створює локальні резервні копії чатів на пристрої, які не використовують наскрізне шифрування (E2EE). Для резервного копіювання у хмарі, наприклад у Google Drive або iCloud, WhatsApp пропонує можливість захисту даних за допомогою E2EE [23].

Однак локальні резервні копії залишаються доступними без наскрізного шифрування, тому у разі крадіжки або втрати пристрою зловмисники можуть отримати доступ до повідомлень. Це підкреслює обмеження E2EE, яке захищає лише дані під час передачі, але не повністю забезпечує конфіденційність на кінцевих пристроях.

2.3.2 Збір метаданих

Хоча WhatsApp застосовує E2EE для захисту вмісту повідомлень, метадані такі як номери телефонів, час доставки, тривалість з'єднань, частота комунікації, розмір цифрового контенту та геолокація – залишаються незашифрованими й зберігаються на серверах компанії. Згідно з політикою WhatsApp, ці дані можуть використовуватися для аналітики та оптимізації сервісу, але вони також дозволяють створювати детальні профілі користувачів. Дослідження демонструють, що метадані можуть розкривати соціальні зв'язки, поведінкові патерни та навіть особисті вподобання, що становить значний ризик для приватності. WhatsApp також вимагає доступу до повного списку контактів користувача, включаючи тих, хто не використовує додаток. Це дозволяє компанії будувати соціальні графи, аналізуючи мережі взаємодій. Відсутність можливості вибіркового додавання контактів посилює проблему, оскільки користувачі

змушені ділитися особистими даними без альтернативи. Такі практики викликають занепокоєння, адже метадані, доступні WhatsApp, можуть бути використані урядовими органами чи зловмисниками для профілювання [24].

Після придбання WhatsApp компанією Facebook (нині Meta) у 2014 році виникли додаткові ризики для конфіденційності. У 2016 році WhatsApp оновив політику конфіденційності, дозволивши передачу даних, зокрема номерів телефонів, до Facebook та інших компаній Meta. Ці дані використовуються для персоналізації реклами на платформах Meta, що суперечить попереднім обіцянкам WhatsApp не ділитися інформацією для маркетингових цілей. Номер телефону, як унікальний ідентифікатор, дозволяє пов’язувати дані з різних джерел – фінансових, медичних, освітніх – створюючи комплексний профіль користувача.

У групових чатах WhatsApp сервер розподіляє повідомлення між учасниками, що полегшує компанії аналіз соціальних взаємодій. Аналіз трафіку, заснований на обсязі повідомлень і метаданих, може розкрити структуру груп і їхню активність. Поєднання цих даних із інформацією з Facebook створює потужний інструмент для профілювання, який може використовуватися для таргетованої реклами чи інших цілей, впливаючи на поведінку користувачів, наприклад, у політичних кампаніях.

2.3.3 Вразливість повторного шифрування

WhatsApp має особливість, яка потенційно може дозволити зловмисникам отримувати доступ до деяких зашифрованих повідомлень. WhatsApp зберігає недоставлені повідомлення на своїх серверах до 30 днів перед їх видаленням. Реалізація протоколу безпеки, використовуваного для наскрізного шифрування (E2EE) у WhatsApp, передбачає генерацію секретних ключів між учасниками діалогу. Однак нові ключі створюються, коли користувач змінює телефон або перевстановлює додаток. Недоставлені повідомлення, які очікували на доставку, поки користувач був офлайн, автоматично перешифровуються та повторно

надсилаються відправником без можливості для відправника перевірити, чи одержувач є саме тією особою, для якої призначено повідомлення [25]. Відправник отримує сповіщення про цю подію лише за умови ввімкнення відповідної функції в налаштуваннях, інакше сповіщення не надсилається, що відрізняє WhatsApp від Signal, оскільки там користувач завжди сповіщається в цьому випадку. Повторне шифрування та відправка раніше недоставлених повідомлень може потенційно дозволити третій стороні перехопити та прочитати недоставлені повідомлення користувача, наприклад, у ситуації, коли зловмисник викрав SIM-картку користувача. Вставивши вкрадену SIM-картку в інший телефон, третя сторона теоретично може отримати доступ до всіх повідомлень, які ще не були доставлені користувачу.

2.3.4 Media File Jacking

Користувачі зазвичай довіряють месенджерам, що використовуюють E2EE, таким як WhatsApp, стосовно захисту цілісносності вмісту повідомлення. Однак, жоден код не застрахований від вразливостей безпеки. Хоча наскрізне шифрування є ефективним механізмом забезпечення цілісності, воно не може гарантувати безпеку, якщо в коді існують вразливості на рівні додатка. Вразливість Media File Jacking, показує, що зловмисники можуть успішно маніпулювати медіафайлами, використовуючи логічні недоліки в додатках.

В деяких випадках цілісність файлів, отриманих через месенджери може бути скомпрометована, і медіафайли WhatsApp можуть бути модифіковані зловмисниками. Дана уразливість системи безпеки, що отримала назву «Викрадення медіафайлів», за замовчуванням реалізується для WhatsApp та Telegram на Android, за умови, що ввімкнено функцію збереження у спільне сховище даних. Уразливість виникає через наявність затримки між тим, коли медіафайли, отримані через месенджери, записуються на диск, та коли вони звідти завантажуються в інтерфейс користувача чату.

Першочергове завантаження медіафайлів у сховище, та подальше їх зчитування звідти створює критичний проміжок часу, в який зловмисники можуть втрутитися та маніпулювати медіафайлами без відома користувача. Використання даної вразливості системи безпеки, дозволяє модифікувати отримані файли, цим самим порушуючи цілісність даних, таких як особисті фотографії та відео, корпоративні та фінансові документи, голосові повідомлення. Зловмисники можуть скористатися довірчими відносинами між відправником та одержувачем, експлуатуючи дану вразливість месенджерів в особистих цілях.

Принцип реалізації вразливості наступний. Android використовує файлову систему, подібну до дискових файлових систем на інших платформах. Система пропонує кілька варіантів збереження даних програм:

App-specific storage: використовується для збереження файлів, призначених для використання лише певною програмою. Файли можуть збрігатися у спеціальних каталогах у внутрішньому томі сховища, або в різних спеціалізованих каталогах у зовнішньому сховищі. Каталоги у внутрішньому сховищі, використовуються для зберігання конфіденційної інформації, до якої інші програми не повинні мати доступ.

Shared storage (спільне сховище): для зберігання файлів, якими певна програма має намір ділитися з іншими програмами, включаючи медіафайли, документи та інші файли [26].

За замовчуванням WhatsApp зберігає файли у спільному сховищі. Той факт, що файли зберігаються у спільному сховищі та завантажуються з нього без належних механізмів безпеки, дозволяє зловмисним програмам, що мають дозвіл на запис у спільне сховище (WRITE_EXTERNAL_STORAGE), модифікувати збережені файли перед їх відкриттям через месенджер. Даний дозвіл є дуже поширеним серед додатків для Android, і користувачі, як правило, не вагаючись надають дозвіл у під час встановлення додатку.

Сам механізм роботи зображено на рис 2.4:

- Користувач отримує файл, і він завантажується у спільне сховище

- Зловмисне ПЗ виявляє зміни в спільному сховищі, і в проміжок часу між завантаженням та читанням медіафайлу через месенджер, зловмисне ПЗ модифікує файл
- Модифікований файл завантажується у месенджер, без сповіщень про порушення його цілісності.

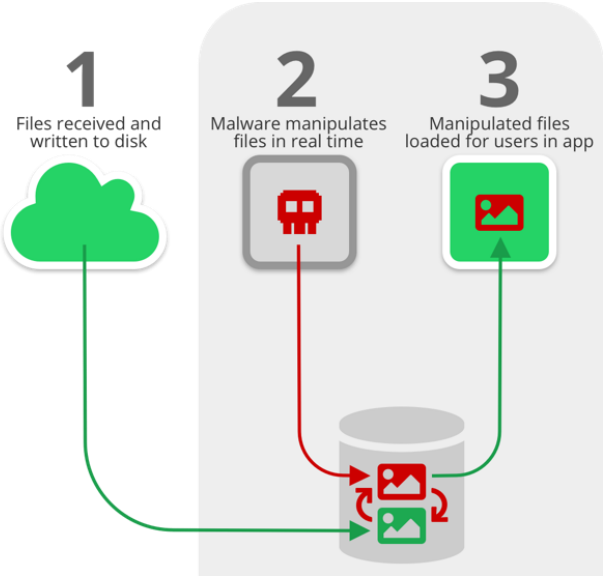

Рисунок 2.4 – Media File Jacking [27]

Вразливість Media File Jacking становить серйозну загрозу безпеці даних у месенджерах, таких як WhatsApp, дозволяючи зловмисникам маніпулювати медіафайлами в реальному часі. Ця атака експлуатує довіру користувачів до отриманих файлів, створюючи ризики для конфіденційності, фінансової безпеки та корпоративної діяльності. Нижче наведено приклади експлуатації розглянутої вразливості.

1) Маніпуляція зображеннями

Шкідливе програмне забезпечення, яке маскується під легітимний додаток, може здійснювати атаку Media File Jacking, змінюючи зображення, отримані через месенджер, у фоновому режимі без відома користувача. Такі програми відстежують медіафайли, що надходять через WhatsApp, і можуть, наприклад, застосовувати технології розпізнавання облич для заміни осіб на зображеннях іншими об’єктами чи особами. Хоча на перший погляд ця маніпуляція може здаватися незначною, вона демонструє потенціал для несанкціонованої зміни

візуального контенту в реальному часі, що може бути використано для дезінформації, шантажу чи порушення приватності

2) Фальсифікація документів

Атака Media File Jacking створює можливість маніпулювання отриманими документами в тому числі і фінансовими. Шкідливий додаток, який виглядає як безпечний, відстежує PDF-файли з рахунками, отриманими через WhatsApp, і програмно змінює реквізити банківського рахунку на дані, контрольовані зловмисником. Користувач отримує документ, який здається легітимним, не підозрюючи про підміну. Така атака може мати масовий характер, якщо зловмисники автоматизують підміну реквізитів у всіх отриманих інвойсах, що загрожує значними фінансовими втратами для користувачів, які використовують месенджери для ділових комунікацій. Виявлення шахрайства зазвичай відбувається із запізненням, коли кошти вже переведено на рахунок зловмисника.

3) Аудіоспуфінг

Вразливість Media File Jacking дозволяє зловмисникам здійснювати аудіоспуфінг, використовуючи довіру між учасниками комунікації, наприклад, у корпоративному середовищі. За допомогою технологій глибокого навчання, які стають дедалі доступнішими, зловмисник може модифікувати аудіоповідомлення, надіслані через WhatsApp, реконструюючи голос відправника. Наприклад, у сценарії, коли генеральний директор надсилає фінансовому директору аудіозапис із проханням виконати певну дію, зловмисник може змінити вміст повідомлення, замінивши його на фальшивий запит про терміновий переказ коштів на шахрайський рахунок. Одержувач, сприймаючи голос як автентичний, може виконати запит, не підозрюючи обману. Ця техніка є особливо ефективною через високу правдоподібність фальсифікованого аудіо, що створює значні ризики для організацій.

2.4 Фішингові атаки

Актуальним прикладом атаки на месенджери є отримання доступу до акаунтів українських військових у Signal, російськими хакерами за допомогою експлуатації функції «підключені пристрої».

Найновіша та найпоширеніша техніка, що лежить в основі спроб російських зловмисників скомпрометувати облікові записи Signal, – це зловживання легітимною функцією застосунку «підключені пристрої», яка дозволяє використовувати Signal на кількох пристроях одночасно.

Для підключення додаткового пристрою зазвичай потрібно сканувати згенерований QR-код, саме це і використали російські хакери для прив’язки власних пристроїв до акаунтів жертв. Зловмисники створили власні шкідливі QR-коди, які після сканування пов’язують обліковий запис жертви з інстансом Signal, керованим зловмисником. Як наслідок успішної експлуатації даної вразливості усі майбутні повідомлення будуть синхронно надсилатися як жертві, так і зловмиснику в режимі реального часу. Дана атака надає зловмиснику засіб прослуховування захищених розмов без необхідності повного компрометування пристрою або акаунту жертви [28].

Щоб скомпрометувати облікові записи Signal за допомогою функції «підключених пристроїв» , один підозрюваний російський шпигунський кластер, відстежений як UNC5792, підробив сторінки «запрошень до групи» замінивши очікуване перенаправлення до групи Signal перенаправленням на шкідливу URL-адресу, створену для зв'язування пристрою, контрольованого зловмисником, з обліковим записом жертви Signal.

У кожному з підроблених групових запрошень код JavaScript, який зазвичай перенаправляє користувача для приєднання до групи Signal, був замінений шкідливим блоком, що містить універсальний ідентифікатор ресурсу (URI), який використовується Signal для зв'язування нового пристрою з Signal, обманом змушуючи жертв пов'язувати свої облікові записи Signal з пристроєм, контрольованим UNC5792.

Приклад фішингової веб-сторінки, яка перенаправляє жертв на вторинну фішингову інфраструктуру, маскуючись під легітимні інструкції щодо зв’язування пристроїв, надані Signal наведено на рис. 2.5.

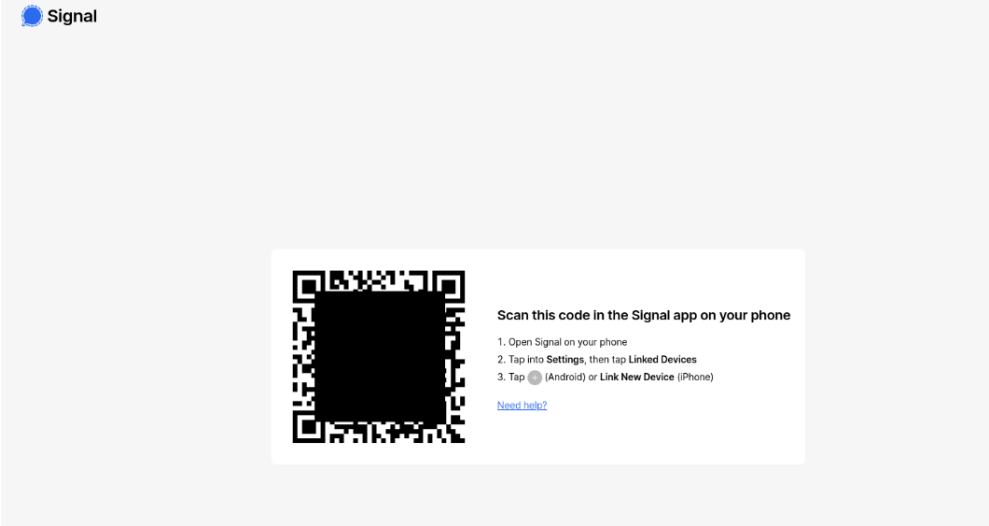

Рисунок 2.5 – Фішингова сторінка зв’язування пристроїв у Signal [28]

Інша пов’язана з росією хакерська група UNC4221 активно атакувала облікові записи Signal, що використовуються українськими військовослужбовцями. Група використовує спеціалізований фішинговий комплект Signal, розроблений для імітації компонентів програми Kropyva, що використовується Збройними Силами України для наведення артилерії. Подібно до підходу соціальної інженерії, який використовує UNC5792, UNC4221 також намагалася замаскувати свою функцію підключення пристроїв під запрошення до групи Signal від довіреного контакту.

На рис. 2.6 наведено приклад фішингу зі шкідливим QR-кодом для зв’язування пристроїв, безпосередньо вбудованим в підроблений програмний застосунок «Кропива».

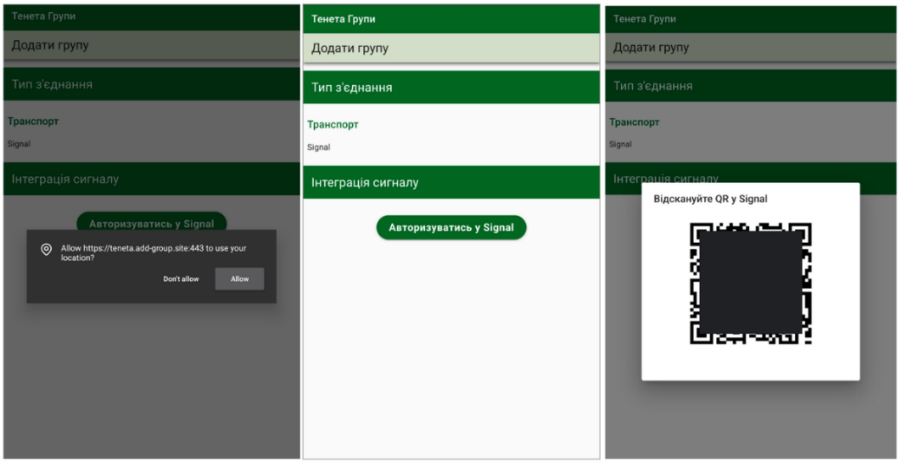

Рисунок 2.6 – Фішингова копія додатка «Кропива» [28]

У попередніх операціях у 2022 році фішингові сторінки UNC4221 були створені так, щоб виглядати як легітимне сповіщення безпеки від Signal (рис. 2.7).

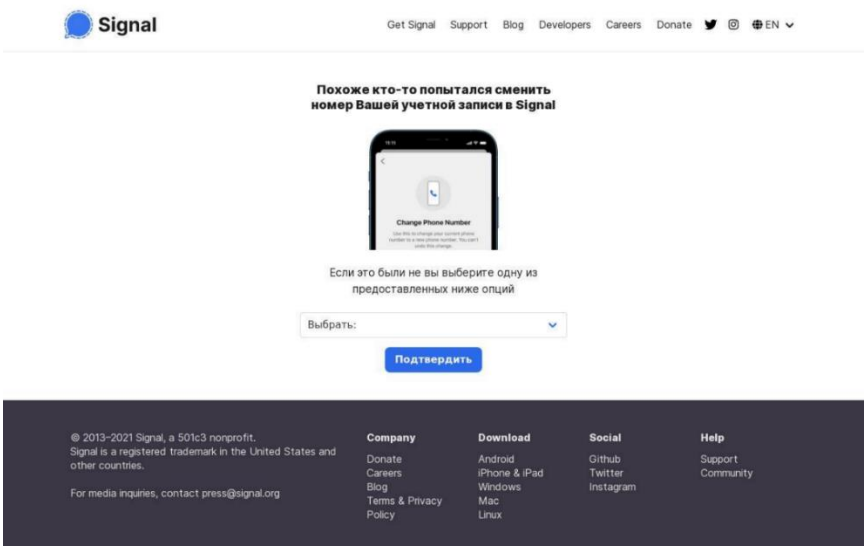

Рисунок 2.7 – Фішингова сторінка сповіщення безпеки Signal [28]

Подібно до атаки на Signal була і компанія по компрометації акаунтів WhatsApp зафіксована урядовою командою реагування на комп'ютерні надзвичайні події України CERT-UA.

Дана атака також експлуатувала функцію «підключені пристрої» месенджеру WhatsApp. Фішинг працював шляхом розповсюдження повідомлень із закликом до голосування за електронну петицію щодо присвоєння звання

«Герой України». Для подальшої реалізації атаки було створено вебресурс, що імітував офіційну сторінку «Електронних петицій».

У разі переходу за посиланням жертві пропонується ввести номер мобільного телефону, отримати згенерований код та використати його для додавання стороннього пристрою до облікового запису WhatsApp в налаштуваннях додатку [29]. На рис. 2.8 наведена покрокова схема реалізації фішингової атаки.

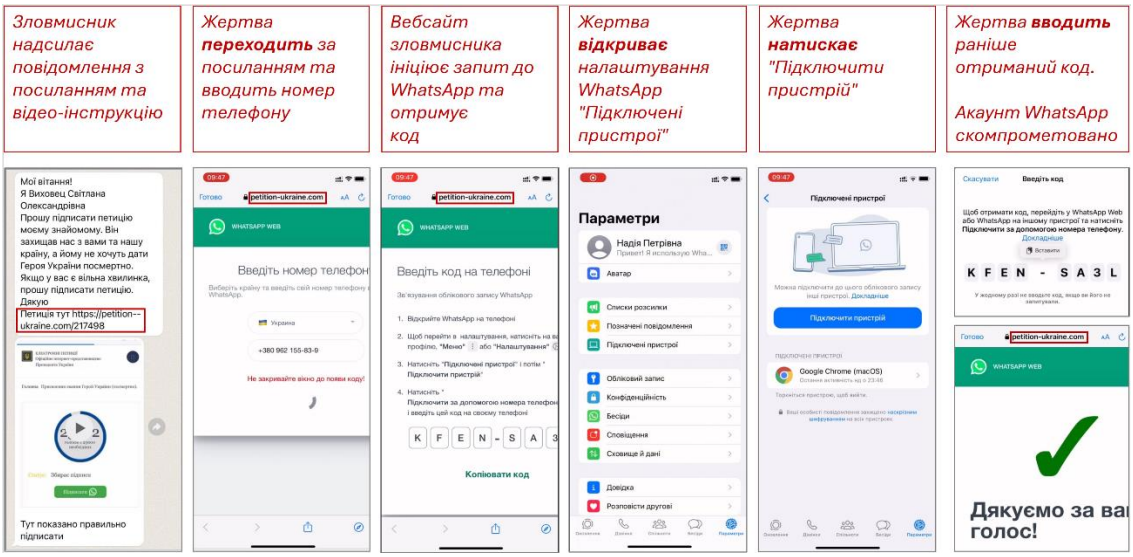

Рисунок 2.8 – реалізація фішингу через електронну петицію [29]

Штатний механізм щодо додавання довіреного пристрою з використанням буквено-цифрового чи QR коду імплементовано в усіх сучасних месенджерах. Тому цей функціонал для може бути широко використаний зловмисниками для отримання несанкціонованого доступу.

Висновки за розділом 2

У 2 розділі досліджено безпеку месенджера WhatsApp, з акцентом на механізми безпеки, такі як наскрізне шифрування (E2EE) на базі протоколу Signal, а також продемонстровано вразливості, які обмежують ефективність цих механізмів. Проведений аналіз демонструє, що, незважаючи на високий рівень

криптографічного захисту, WhatsApp залишається вразливим до низки загроз, що вимагає впровадження додаткових рішень захисту.

Наскрізне шифрування, реалізоване через протокол Signal, є основою безпеки WhatsApp. Протокол поєднує Extended Triple Diffie-Hellman (X3DH) для ініціації сеансу та Double Ratchet для динамічного оновлення ключів, використовуючи криптографічні алгоритми Curve25519, AES-256 і HMAC-SHA256. Ці механізми забезпечують конфіденційність, цілісність, автентифікацію, forward secrecy, post-compromise security. E2EE гарантує, що вміст повідомлень доступний лише відправнику та одержувачу, унеможливлюючи доступ сторонніх сторін, включно з провайдерами послуг. Проте захист обмежується вмістом повідомлень, залишаючи метадані (номери телефонів, часові мітки) незашифрованими. Ці дані, зібрані WhatsApp і передані до екосистеми Meta, дозволяють створювати детальні профілі користувачів, що становить ризик для приватності, особливо в контексті державного стеження чи комерційного профілювання.

У WhatsApp є вразливості, від яких не захищає наскрізне шифрування. Вразливість повторного шифрування (re-encryption vulnerability) дозволяє перехоплення недоставлених повідомлень у разі компрометації SIM-картки, оскільки нові ключі генеруються без обов’язкової верифікації одержувача. Media File Jacking експлуатує затримку між збереженням і відображенням медіафайлів у спільному сховищі Android, дозволяючи зловмисникам маніпулювати зображеннями, фінансовими документами та аудіоповідомленнями, що загрожує дезінформацією, фінансовими втратами та корпоративною безпекою. Фішингові атаки, такі як експлуатація функції «підключені пристрої» в Signal і WhatsApp, демонструють вразливість до соціальної інженерії, коли зловмисники обманом отримують доступ до облікових записів через підроблені QR-коди чи фішингові вебресурси.

Ці вразливості підкреслюють обмеження стандартних механізмів безпеки WhatsApp, які не можуть повною мірою захистити користувачів від атак, спрямованих на кінцеві пристрої чи загроз спричинених людським фактором.

Збір метаданих і їх інтеграція з платформами Meta посилюють загрози профілювання, тоді як вразливості на рівні додатка, такі як Media File Jacking, дозволяють маніпулювати даними, обходячи E2EE.

Незважаючи на те, що WhatsApp вважається одними із найбільш надійних месенджерів завдяки імплементації наскрізного шифрування (E2EE), він все ще не захистить дані у разі фішингу. У такому разі, розробка механізму шифрування трафіку для передачі даних через месенджери забезпечить підвищений рівень безпеки та конфіденційності комунікацій.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Вибір мови програмування

C# є потужною об’єктно-орієнтованою мовою програмування загального призначення, розробленою компанією Microsoft яка поєднує в собі гнучкість і підтримку сучасних парадигм програмування із простотою використання. Мова C# є мовою для платформи .NET, безкоштовного, кросплатформного середовища розробки з відкритим вихідним кодом.

C# дозволяє реалізовувати принципи ООП, такі як інкапсуляція, спадкування та поліморфізм, що є важливим для створення модульної архітектури. Використання інтерфейсів дозволить притримуватися принципів SOLID для абстрагування логіки шифрування та логіки взаємодії з месенджером від роботи з графічним інтерфейсом, що забезпечить гнучкість для потенційної зміни обраного месенджера чи алгоритма шифрування. Використання асинхронного програмування є необхідним для ефективної взаємодії з месенджером, оновлення та розшифрування повідомлень з чату без блокування інтерфейсу користувача. Сильна статична типізація C# зменшує ризик помилок під час виконання криптографічних операцій, а автоматичне управління пам’яттю знижує ризики безпеки та забезпечує коректне звільнення криптографічних ресурсів.

Платформа .NET включає широкий спектр бібліотек, зокрема для криптографії (System.Security.Cryptography), що забезпечує реалізацію надійних алгоритмів шифрування таких як AES-256 та алгоритму хешування SHA-256, що будуть використані для шифрування повідомлень. .NET підтримує технологію Windows Presentation Foundation (WPF) для створення сучасного графічного інтерфейсу з підтримкою динамічного оновлення даних, що є важливим для

коректного відображення повідомлень у чаті месенджера, та їх постійного оновлення.

Таким чином, підтримка принципів SOLID через об’єктно-орієнтоване програмування, використання інтерфейсів та абстракцій, автоматичне управління пам’яттю та строга типізація, наявність сучасних криптографічних бібліотек та підтримка WPF для реалізації відповідного інтерфейсу користувача, забезпечують модульність, безпеку та ефективність, необхідні для реалізації механізму додаткового шифрування повідомлень у месенджерах, що робить вибір мови програмування C# та платформи .NET оптимальним рішенням.

3.2 Реалізація взаємодії з месенджерами

Більшість месенджерів не мають відкритого API, але водночас підтримують браузерні версії, які дозволяють користувачам відправляти та отримувати повідомлення через веб-інтерфейс. В розроблюваному застосунку взаємодія з обраним месенджером (WhatsApp) відбувається через відкриття його web версії та подальшу роботу з ним за допомогою Selenium WebDriver.

SeleniumWebDriver – це інструмент для автоматизації роботи у веб-браузері. Він є частиною набору інструментів Selenium, призначених для тестування веб-інтерфейсів та автоматизації взаємодії з веб-сервісами. WebDriver використовує API автоматизації роботи з браузером, що надаються вендорами, для керування браузером та проведення тестування. Він підтримує різні браузери, зокрема Google Chrome, Mozilla Firefox, Microsoft Edge та інші, завдяки драйверам, спецефічним для кожного браузера. Selenium WebDriver підтримує декілька мов програмування, такі як Java, Python, JavaScript, і зокрема C#, що ідеально підходить для розроблюваного проекту [30].

WebDriver надає можливість пошуку необхідних елементів веб-сторінки та взаємодії з ними за допомогою, наприклад, CSS-селекторів та XPath, що дозволяє імітувати дії користувача, такі як копіювання і введення тексту, натискання кнопок.

Таким чином, SeleniumWebDriver є потужним інструментом для автоматизації взаємодії з веб-інтерфейсами, здатним до крос-браузерної роботи. Його гнучкість у взаємодії з динамічними веб-елементами, здатність знаходити та взаємодіяти з необхідними елементами веб-сторінки, дозволять частково компенсувати відсутність відкритих API для взаємодії з більшістю месенджерів, що робить його вдалим рішенням для реалізації розроблюваного застосунку.

3.3 Архітектура застосунку

Архітектура застосунку базується на чіткому розділенні бізнес-логіки, логіки шифрування та інтерфейсу користувача. Використання інтерфейсів та ізоляції логіки забезпечує відповідність принципам SOLID, зокрема принципу єдиної відповідальності (SRP), принципу відкритості/закритості (OCP), та інверсії залежностей (DIP), що робить архітектуру придатною до розширення та дозволяє легко адаптувати систему до інших месенджерів та алгоритмів шифрування у майбутньому.

IMessenger.cs і *IEncryption.cs* визначають інтерфейси, які є ключовими для забезпечення модульності та слабкого зв’язування між компонентами. Інтерфейс IMessenger декларує методи для роботи з месенджером, такі як відправлення повідомлень, читання чату та обробка файлів, тоді як IEncryption визначає методи шифрування й дешифрування. Використання інтерфейсів дозволяє MainWindow залежати від абстракцій, а не конкретних класів (WhatsAppMessenger чи AesEncryption). Це забезпечує гнучкість для заміни реалізації месенджера чи алгоритму шифрування без модифікації UI-логіки.

MainWindow.xaml.cs є центральною частиною UI-шару застосунку, у ньому реалізовано логіку графічного інтерфейсу. Він відповідає за взаємодію з користувачем, включаючи обробку подій кнопок, для надсилання та читання повідомлень і файлів, вибору контакту та введення паролю шифрування. Клас MainWindow ініціалізує залежності IWebDriver (використання Selenium

WebDriver для взаємодії з WhatsApp), IEncryption (для реалізації алгоритмів шифрування) та IMessenger (для роботи з обраним месенджером).

AesEncryption.cs відповідає за криптографічні операції, містить клас AesEncryption що реалізує логіку шифрування і дешифрування для повідомлень та файлів алгоритмом шифрування AES. Клас AesEncryption імплементує інтерфейс IEncryption, що включає методи Encrypt, Decrypt, EncryptFile, DecryptFile та GenerateKeys.

MessengerService.cs відповідає за логіку взаємодії з месенджером, містить клас WhatsAppMessenger, який реалізує інтерфейс IMessenger і відповідає за взаємодію з WhatsApp через Selenium WebDriver. Цей клас інкапсулює бізнес-логіку, пов’язану з відправленням і читанням повідомлень, а також обробкою файлів. Методи OpenChat, SendMessage, ReadMessages, GetMessagesAsync, SendEncryptedFile і SaveDecryptedFile ізольовані від UI і працюють із зашифрованими даними через IEncryption.

3.4 Реалізація шифрування

Для реалізації додаткового захисту даних при передачі через месенджери в розроблюваному додатку, було обрано алгоритм симетричного шифрування AES. AES є сучасним стійким симетричним алгоритмом блочного шифрування прийнятим Національним інститутом стандартів і технологій США (NIST). Зокрема, використання випадкового вектора ініціалізації (IV) у режимі CBC (Cipher Block Chaining) забезпечує унікальність шифротектсу для однакових повідомлень, що підвищує стійкість до атак. Для реалізації алгоритму шифрування AES використовується бібліотека System.Security. Cryptography. Для реалізації функціоналу програми даний клас містить п’ять методів.

```
2 references
public void GenerateKeys(string password)
{
    var hash = SHA256.HashData(Encoding.UTF8.GetBytes(password));
    _key = hash;
}
```

Рисунок 3.1 – метод GenerateKeys

Метод GenerateKeys (рис. 3.1) реалізує хешування, введеного користувачем паролю, за допомогою алгоритма SHA-256 для отримання 256 бітного ключа, що буде в подальшому використаний для обраного алгоритму шифрування AES-256.

```
public byte[] Encrypt(string plaintext)
{
    if (plaintext == null || plaintext.Length <= 0)
        throw new ArgumentNullException("cipherText");
    if (_key == null || _key.Length <= 0)
        throw new ArgumentNullException("Key");

    using var aesAlg = Aes.Create();
    aesAlg.Key = _key;
    aesAlg.GenerateIV(); // випадковий IV для кожного шифрування

    var encryptor = aesAlg.CreateEncryptor();
    using var msEncrypt = new MemoryStream();

    // IV  записується на початок
    msEncrypt.Write(aesAlg.IV, 0, aesAlg.IV.Length);

    using (var csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
    {
        var plainBytes = Encoding.UTF8.GetBytes(plaintext);
        csEncrypt.Write(plainBytes, 0, plainBytes.Length);
        csEncrypt.FlushFinalBlock();
    }

    return msEncrypt.ToArray(); // [IV | CipherText]
}
```

Рисунок 3.2 – метод Encrypt

Метод Encrypt (рис. 3.2) реалізує симетричне шифрування вхідних повідомлень з використанням алгоритму AES. Метод починається з перевірки наявності вхідного повідомлення та вже згенерованого ключа. Метод Aes.Create() ініціалізує новий екземпляр AES з налаштуваннями за замовчуванням (розмір ключа 256 біт, розмір блоку 128 біт, режим шифрування CBC, схема доповнення блоку PKCS7). Створюється encryptor та MemoryStream для подальшої роботи.

Для досягнення семантичної безпеки при кожній операції шифрування генерується новий випадковий вектор ініціалізації (IV). Згенерований вектор ініціалізації одразу записується на початок об’єкта MemoryStream. Далі за допомогою потоку CryptoStream з використанням енкриптора (encryptor) у режимі запису (CryptoStreamMode.Read) введене повідомлення шифрується і записується в MemoryStream. Метод FlushFinalBlock() відповідає за коректне завершення операції шифрування та доповнення останнього блоку шифрування (pading). В результаті метод Encrypt() повертає масив байтів де перші 16 байтів містять вектор ініціалізації, а далі йде зашифроване повідомлення.

```
public string Decrypt(byte[] ciphertext)
{
    if (ciphertext == null || ciphertext.Length <= 0)
        throw new ArgumentNullException("cipherText");
    if (_key == null || _key.Length <= 0)
        throw new ArgumentNullException("Key");

    using var aesAlg = Aes.Create();
    aesAlg.Key = _key;

    // Виділяємо IV з перших 16 байтів
    var iv = ciphertext.Take(16).ToArray();
    aesAlg.IV = iv;

    var actualCipher = ciphertext.Skip(16).ToArray();
    var decryptor = aesAlg.CreateDecryptor();

    using var msDecrypt = new MemoryStream(actualCipher);
    using var csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read);
    using var msPlain = new MemoryStream();
    csDecrypt.CopyTo(msPlain);
    return Encoding.UTF8.GetString(msPlain.ToArray());
}
```

Рисунок 3.3 – метод Decrypt

Метод Decrypt (рис. 3.3) приймає байтовий масив шифротексту. Перевіряє коректність вхідних параметрів та наявність згенерованого криптографічного ключа. При дешифруванні перші 16 байт повідомлення записуються як IV, а подальші інтерпретуються як повідомлення. Для розшифрування об’єкт CryptoStream, використовується з дескриптором (descryptor) у режимі читання (CryptoStreamMode.Read). На завершальному етапі вміст MemoryStream перетворюється у рядок за допомогою декодування UTF-8.

```
    using var aesAlg = Aes.Create();
    aesAlg.Key = _key;
    aesAlg.GenerateIV();

    using var msEncrypt = new MemoryStream();
    msEncrypt.Write(aesAlg.IV, 0, aesAlg.IV.Length);

    var encryptor = aesAlg.CreateEncryptor();
    using (var csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
    using (var fsInput = File.OpenRead(filePath))
    {
        fsInput.CopyTo(csEncrypt);
        csEncrypt.FlushFinalBlock();
    }

    return msEncrypt.ToArray();
}
```

Рисунок 3.4– Основна частина методу EncryptFile

```
    using var aesAlg = Aes.Create();
    aesAlg.Key = _key;

    var iv = encryptedData.Take(16).ToArray();
    aesAlg.IV = iv;

    var actualCipher = encryptedData.Skip(16).ToArray();
    var decryptor = aesAlg.CreateDecryptor();

    using var msDecrypt = new MemoryStream(actualCipher);
    using var csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read);
    using var fsOutput = File.Create(outputFilePath);
    csDecrypt.CopyTo(fsOutput);
}
```

Рисунок 3.5 – Основна частина методу DecryptFile

Симетричне шифрування та розшифрування файлів алгоритмом AES реалізується в методах EncryptFile (string File.Path) (рис. 3.4) та DecryptFile (byte[] encryptedData, string outputFilePath) (рис. 3.5). Дані методи працюють подібно до шифрування повідомлень, використовують AES в режимі CBC та генерують випадковий IV для кожного файлу. Основною відмінністю даних методів є реалізація роботи з файлами, а саме читання та запис файлів з отриманих відповідно string filePath та string outupFilePath.

3.5 Робота застосунку

Після запуску програми у конструкторі MainWindow (рис. 3.6) ініціалізується WPF-інтерфейс, створюється ChromeDriver для керування браузером, відкривається сторінка WhatsApp, і налаштовується WebDriverWait для очікування елементів вебсторінки. Також створюються об’єкти

AesEncryption для шифрування та WhatsAppMessenger для роботи з месенджером.

```
0 references
public MainWindow()
{
    InitializeComponent();
    _driver = new ChromeDriver();
    _driver.Navigate().GoToUrl("https://web.whatsapp.com");
    _wait = new WebDriverWait(_driver, TimeSpan.FromSeconds(10));
    _encryption = new AesEncryption();
    _messenger = new WhatsAppMessenger(_driver, _wait, _encryption);

    MessageBox.Show("Відскануйте QR-код для входу в WhatsApp");
}
```

Рисунок 3.6 – MainWindow

З використанням Selenium Web Drivers відкривається WhatsApp Web (web.whatsapp.com) у браузері. На рис. 3.7 зображено вікно входу у WhatsApp Web, де користувач має просканувати QR для входу у свій акаунт за допомогою функції devices linking, або ввійти за номером телефону.

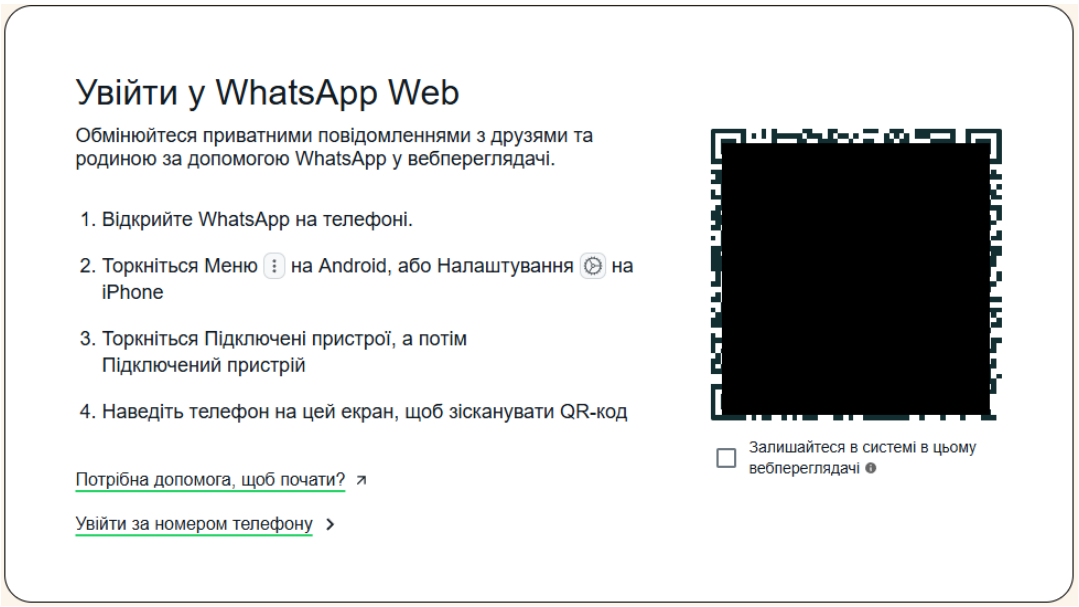

Рисунок 3.7 – Відкриття web.whatsapp.com

Після входу можна перейти до головного вікна програми (рис. 3.8). Необхідно ввести ім’я контакту з яким буде відкрито чат.

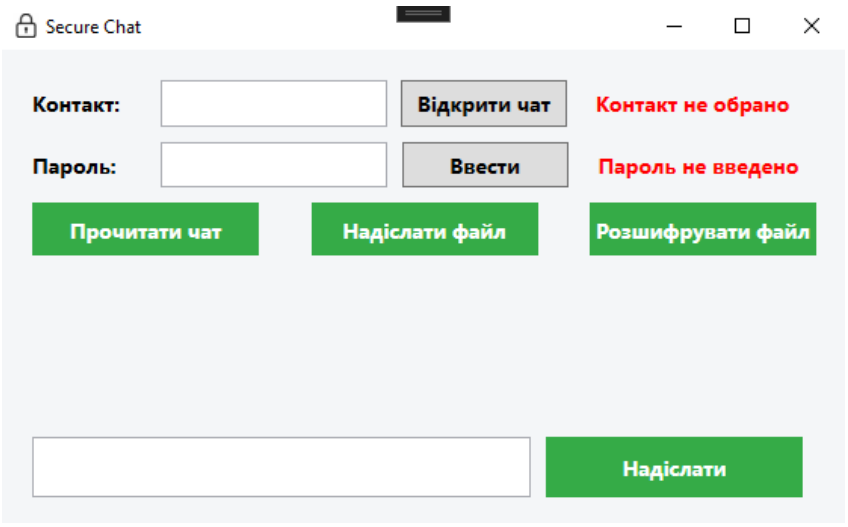

Рисунок 3.8 – Головне вікно програми

Для відкриття чату використовується метод OpenChat (рис. 3.9) класу WhatsApp. Метод OpenChat із WhatsAppMessenger викликається через _messenger.OpenChat при введенні імені контакту. Він використовує Selenium для пошуку поля введення через XPath, очищає його, вводить ім’я контакту, чекає 2 секунди поки WhatsApp шукає вказаний контакт, далі знаходить контакт через CssSelector та натискає на нього, відкриваючи чат. Цей метод ізолює логіку взаємодії з месенджером, дозволяючи легко адаптувати її для інших платформ.

```
2 references
public void OpenChat(string contactName)
{
    var searchBox = _wait.Until(d => d.FindElement(
        By.XPath("//div[@contenteditable='true']")));
    searchBox.SendKeys(Keys.Control + "a");
    searchBox.SendKeys(Keys.Delete);
    searchBox.SendKeys(contactName);
    Thread.Sleep(2000);
    var contact = _wait.Until(d => d.FindElement(
        By.XPath($"//span[@title='{contactName}']")));
    contact.Click();
}
```

Рисунок 3.9 – метод OpenChat

Далі необхідно ввсети пароль який буде використовуватись для шифрування та розшифрування повідомлень, після чого буде згенеровано ключ методом GenerateKeys.

Після натискання на кнопку «Прочитати повідомлення» метод ReadMessages із WhatsAppMessenger викликається в GetMessagesAsync. Він асинхронно читає повідомлення з WhatsApp через Selenium, витягує текст час відправлення і дані відправника, розшифровує повідомлення за допомогою IEncryption і формує рядок із розшифрованими даними. Незашифровані повідомлення пропускаються, що забезпечує коректну обробку чату. На рис. 3.10 зображено розшифрований чат для контакту “Test”.

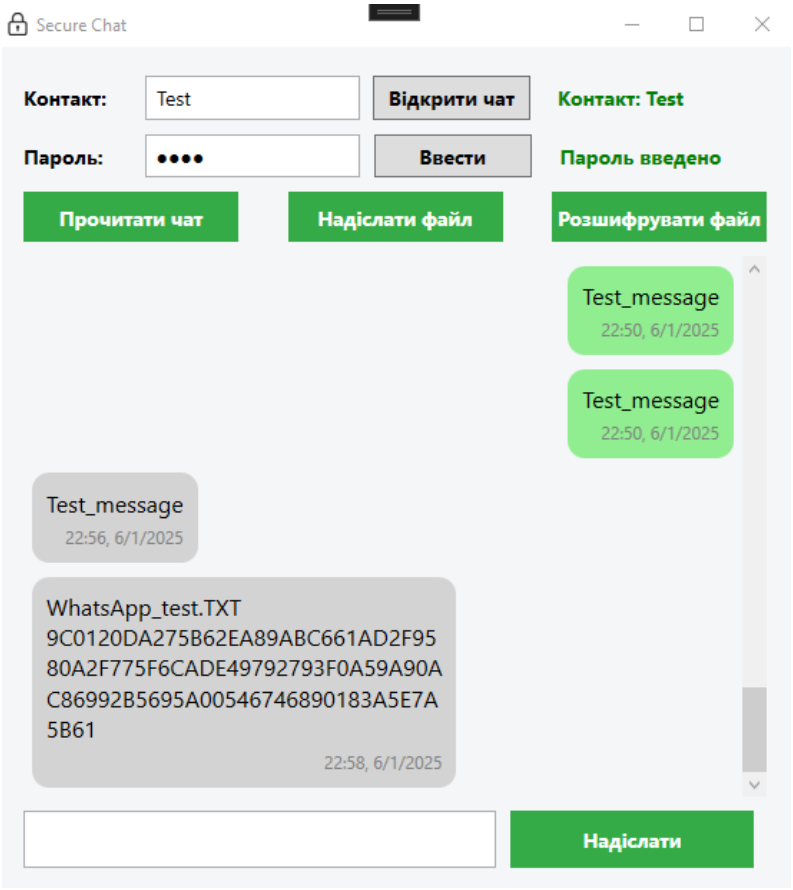

Рисунок 3.10 – Прочитаний чат

Метод StartMessageRefreshLoop (рис. 3.11) ініціалізує CancellationTokenSource для керування циклом і виконує асинхронний цикл, який перевіряє, чи не скасовано токен. Якщо _messenger ініціалізовано, викликається _messenger.GetMessagesAsync для отримання форматованих повідомлень у вигляді MessageItem, для їх коректного відображення в чаті. Через Dispatcher.Invoke список MessagesListBox очищається, і до нього додаються нові

повідомлення. Оновлення повідомлень викликаєтсья кожні 5 секунд за допомогою Task.Delay. Цей метод ізолює UI-логіку, передаючи обробку повідомлень до бізнес-логіки в WhatsAppMessenger.

```
1 reference
private async void StartMessageRefreshLoop()
{
    _cts = new CancellationTokenSource();
    try
    {
        while (!_cts.Token.IsCancellationRequested)
        {
            if (_messenger != null)
            {
                var messageItems = await _messenger.GetMessagesAsync();
                MessagesListBox.Dispatcher.Invoke(() =>
                {
                    MessagesListBox.Items.Clear();
                    foreach (var item in messageItems)
                    {
                        MessagesListBox.Items.Add(item);
                    }
                });
            }
            await Task.Delay(TimeSpan.FromSeconds(5), _cts.Token);
        }
    }
    catch (TaskCanceledException)
    {
        // Цикл зупиняється при закритті програми
        // методом StopMessageRefreshLoop(), що викликаєтсья у OnClosed
    }
}
```

Рисунок 3.11 – метод StartMessageRefreshLoop

Метод ReadMessages (рис. 3.12) із WhatsAppMessenger використовує Selenium WebDriver для пошуку всіх повідомлень у чаті WhatsApp за CSS-селектором. Для кожного повідомлення він отримує дані відправника через атрибут data-pre-plain-text. Отримані зашифровані повідомлення (рис. 3.13) розшифровуються за допомогою _encryption.Decrypt. Розшифровані повідомлення додаються до результату у форматі «[час] відправник: текст». Незашифровані, або зашифровані іншим ключем повідомлення пропускаються, що забезпечує коректність читання розшифрованого чату. Метод повертає рядок із розшифрованими повідомленнями, який потім обробляється GetMessagesAsync.

```
2 references
public async Task<string> ReadMessages()
{
    var messages = _driver.FindElements(By.CssSelector("span.selectable-text"));
    StringBuilder sb = new StringBuilder();

    foreach (var message in messages)
    {
        var parentElement = message.FindElement(By.XPath("..")).FindElement(By.XPath(".."));
        string sender = parentElement.GetAttribute("data-pre-plain-text");
        string messageText = message.Text;
        string output;

        try
        {
            byte[] encryptedBytes = Convert.FromBase64String(messageText);
            string decryptedMessage = _encryption.Decrypt(encryptedBytes);

            // Незашифровані повідомлення пропускаються
            if (!string.IsNullOrWhiteSpace(decryptedMessage))
            {
                sb.AppendLine($"{sender} {decryptedMessage}");
            }
        }
```

Рисунок 3.12 – Основна частина методу ReadMessages

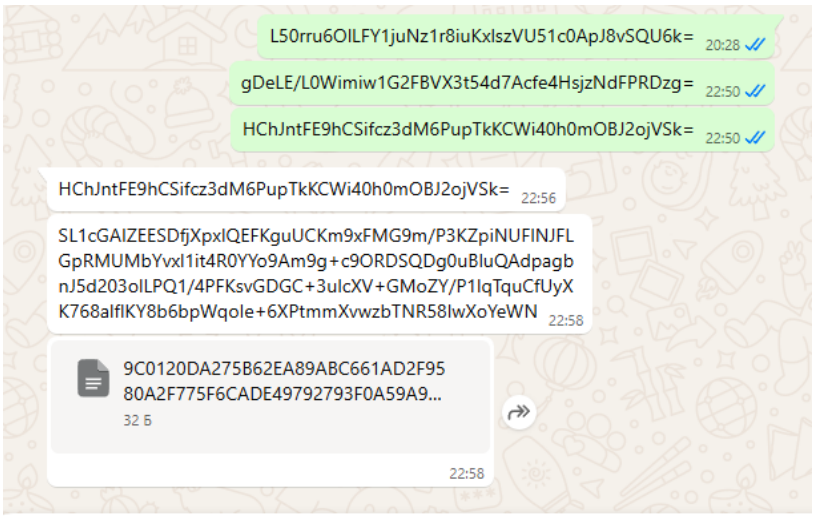

Рисунок 3.13 – Вигляд зашифрованих повідомлень

Метод GetMessagesAsync (рис. 3.14) із WhatsAppMessenger викликає ReadMessages для отримання сирих даних чату, розбиває їх на рядки та обробляє кожен рядок за допомогою регулярного виразу. Він витягує час, відправника та текст повідомлення, створюючи об’єкт MessageItem із відповідним вирівнюванням (праворуч відправлені повідомлення, ліворуч отримані повідомлення) і кольором (зелений для відправлених, сірий для отриманих). Цей метод ізолює логіку форматування, повертаючи структуровані дані для UI, що відповідає принципу єдиної відповідальності (SRP).

```
2 references
public async Task<IEnumerable<MessageItem>> GetMessagesAsync()
{
    var messageItems = new List<MessageItem>();
    string newMessages = await ReadMessages();
    var lines = newMessages.Split(new[] { "\r\n", "\n" }, StringSplitOptions.RemoveEmptyEntries);

    foreach (var rawLine in lines)
    {
        string timePart = "", sender = "", message = "";
        var match = Regex.Match(rawLine, @"\[(.*?)\]\s+(.*?):\s+(.*)");

        timePart = match.Groups[1].Value;
        sender = match.Groups[2].Value;
        message = match.Groups[3].Value;

        bool isMe = sender.Equals("Ps", StringComparison.OrdinalIgnoreCase);
        messageItems.Add(new MessageItem
        {
            Text = message,
            Time = timePart,
            Alignment = isMe ? HorizontalAlignment.Right : HorizontalAlignment.Left,
            BubbleColor = isMe ? Brushes.LightGreen : Brushes.LightGray
        });
    }

    return messageItems;
}
```

Рисунок 3.14 – GetMessagesAsync

Надсилання повідомлення реалізується методом SendMessage (рис. 3.15) класу WhatsAppMessenger. Метод SendMessage приймає текстовий параметр message, який є повідомленням, уведеним користувачем у текстове поле інтерфейсу WPF (SendTextBox). Отримане повідомлення шифрується за допомогою об’єкта _encryption, який імплементує інтерфейс IEncryption (в даному випадку використовується клас AesEncryption), що дозволяє модульно змінювати алгоритми шифрування. Використовуючи Selenium WebDriver, за допомогою CssSelector знаходиться поле введення, куди вставляється та відправляється закодоване у Base64 зашифроване повідомлення.

```
public void SendMessage(string message)
{
    var encryptedMessage = _encryption.Encrypt(message);
    string base64Message = Convert.ToBase64String(encryptedMessage);

    var messageBox = _wait.Until(d => d.FindElement(By.CssSelector("div[contenteditable='true']" +
        "[data-tab='10']")));
    messageBox.SendKeys(base64Message);
    messageBox.SendKeys(Keys.Enter);
}
```

Рисунок 3.15 – метод SendMessage

При натисканні на кнопку відправлення файлів метод SendFileButton_Click відкриває діалогове вікно, дозволяючи користувачу обрати файл, та передає шлях файлу до методу SendEncryptedFile.

Відправка зашифрованих файлів реалізується методом SendEncryptedFile (рис. 3.16) із WhatsAppMessenger. Він шифрує файл і його ім’я за допомогою IEncryption, зберігає зашифрований файл із HEX-іменем (оскільки Base64 містить заборонені символи для використання у іменах файлів) у тимчасовій папці, копіює його в буфер обміну, вставляє в чат WhatsApp та відправляє через Selenium WebDriver. Для того, щоб відправлені та отримані файли були видимими у вікні розроблюваного застосунку, його оригінальна назва та зашифрована назва у HEX окремо шифруються та відправляються текстовим повідомленням.

```
public void SendEncryptedFile(string originalFilePath)
{
    try
    {
        byte[] encryptedData = _encryption.EncryptFile(originalFilePath);
        string fileName = System.IO.Path.GetFileName(originalFilePath);
        var encFileBytes = _encryption.Encrypt(fileName);
        string encryptedFileName = Convert.ToHexString(encFileBytes);
        string tempPath = System.IO.Path.Combine(System.IO.Path.GetTempPath(), encryptedFileName);
        File.WriteAllBytes(tempPath, encryptedData);
        StringCollection fileCollection = new StringCollection();
        fileCollection.Add(tempPath);
        Clipboard.SetFileDropList(fileCollection);
        var messageBox = _wait.Until(d => d.FindElement(By.CssSelector("div[contenteditable='true']" +
            "[data-tab='10']")));
        messageBox.Click();
        Thread.Sleep(500);
        messageBox.SendKeys(Keys.Control + "v");
        Thread.Sleep(1500);
        var sendButton = _wait.Until(d => d.FindElement(By.CssSelector("div[role='button'] " +
            "span[data-icon='wds-ic-send-filled']"))).FindElement(By.XPath(".."));
        sendButton.Click();
        string fileFullName = $"{fileName} {encryptedFileName}";
        SendMessage(fileFullName);
    }
```

Рисунок 3.16 – метод SendEncryptedFile

Для читання файлу, користувач має власноруч завантажити його. При натисканні на кнопку «Розшифрувати файл», відкриваєтсья вікно для вибору зашифрованого файлу, і його шлях передається до методу SaveDecryptedFile.

Метод SaveDecryptedFile (рис. 3.17) читає зашифрований файл (encryptedData), визначає його директорію та HEX-кодовану назву (encryptedfileName). Назва розшифровується через _encryption.Decrypt, перетворюючи HEX у байти та отримуючи оригінальну назву

(decryptedfileName). Зашифрований вміст файлу розшифровується за допомогою _encryption.DecryptFile, який витягує IV із перших 16 байтів і застосовує AES для відновлення оригінального вмісту. Розшифрований файл зберігається з префіксом "decrypted_" у тій самій директорії (рис. 3.18).

```
public string SaveDecryptedFile(string encryptedPath)
{
    try
    {
        byte[] encryptedData = File.ReadAllBytes(encryptedPath);
        string dir = System.IO.Path.GetDirectoryName(encryptedPath);

        // Розшифровка імені HEX

        string encryptedfileName = System.IO.Path.GetFileName(encryptedPath);
        string decryptedfileName = _encryption.Decrypt(Convert.FromHexString(encryptedfileName));

        string decryptedPath = System.IO.Path.Combine(dir, "decrypted_" + decryptedfileName);

        _encryption.DecryptFile(encryptedData, decryptedPath);

        return decryptedPath;
    }
    catch (Exception ex)
    {
        throw new InvalidOperationException($"Помилка розшифування файлу: {ex.Message}", ex);
    }
}
```

Рисунок 3.17 – метод SaveDecryptedFile

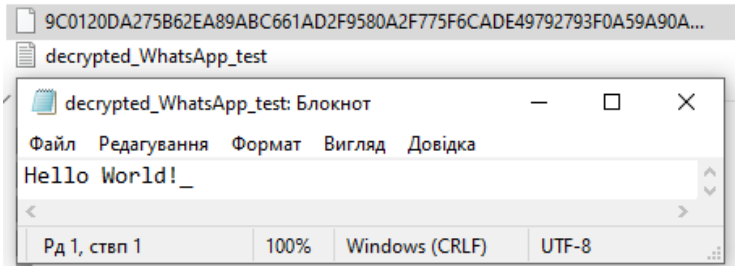

Рисунок 3.18 – Результат розшифрування файлу

Висновки за розділом 3

У третьому розділі було описано процес розробки та роботу програмного застосунку, що реалізує механізм шифрування трафіку для передачі даних через месенджери. Вибір мови програмування C# та платформи .NET обґрунтовано їхньою підтримкою об’єктивно орієнтованого програмування, принципів SOLID та асинхронного програмування, що забезпечує модульність, безпеку та ефективність. У зв’язку з відсутністю відкритого API для більшості популярних месенджерів, в тому числі WhatsApp, взаємодія з месенджером реалізована через

використання його веб-версії. Для автоматизації взаємодії з браузером використано Selenium WebDriver.

Архітектура застосунку побудована на чіткому розділенні UI та бізнес логіки для взаємодії з месенджером і реалізації криптографії за допомогою інтерфейсів IMessenger та IEncryption. Це забезпечує гнучкість для потенційних розширень, таких як підтримка інших месенджерів чи алгоритмів шифрування. У застосунку використовується алгоритм шифрування AES-256 з режимом CBC та генерацією випадкового IV для кожного повідомлення.

Розроблений функціонал реалізує шифрування та дешифрування текстових і файлових повідомлень та їх відправлення через WhatsApp, асинхронне читання чату та його оновлення і зручне подання у інтерфейсі користувача.

Розроблене програмне рішення ефективно захищає від розглянутих у 2 розділі вразливостей. Додатковий шар шифрування у вигляді AES-256 забезпечує конфіденційність даних навіть у разі компрометації пристрою, або акаунту користувача. Модульність архітектури застосунку дозволяє легке розширення кількості підтримуваних месенджерів та алгоритмів шифрування.

ВИСНОВКИ

У даній кваліфікаційній роботі було розроблено програмний застосунок, що дозволяє реалізувати механізм додаткового шифрування даних при передачі у месенджерах

У першому розділі було проведено аналіз актуальності захисту конфіденційності в месенджерах, особливо в умовах воєнного часу в Україні, де вони відіграють критичну роль у комунікаціях. Визначено основні загрози, та механізми захисту, Проведено порівняння рівня безпеки популярних месенджерів. Signal виявився найбезпечнішим завдяки відкритому коду та мінімальному збору метаданих, тоді як WhatsApp, попри використання протоколу Signal, має ризики через інтеграцію з екосистемою Meta.

У другому рохділі було досліджено безпеку WhatsApp, основною перевагою якого є обов’язкове впровадження E2EE, реалізованого через протокол Signal, який використовує алгоритми Curve25519, AES-256 і HMAC-SHA256 для забезпечення конфіденційності та цілісності даних. Однак, не зважаючи на безпеку E2EE, незакритими залишаються вразливості кінцевих пристроїв та ризики пов’язані із соціальною інженерією. Серед оглянутих вразивостей є повторне шифрування (re-encryption vulnerability), Media File Jacking, фішингові атаки та відсутність шифрування попередніх листувань на пристрої користувача. Все це підкреслює необхідність додаткового шифрування для захисту від атак на кінцеві пристрої та соціальної інженерії.

У третьому розділі описано розробку програмного застосунку для шифрування трафіку в месенджерах. Обґрунтовано вибір C# і .NET через підтримку SOLID, асинхронного програмування та криптографічних бібліотек. Взаємодія з WhatsApp реалізована через Selenium WebDriver, компенсуючи відсутність відкритого API. Архітектура застосунку, побудована на інтерфейсах IMessenger і IEncryption, забезпечує модульність і гнучкість. Використання

алгоритму AES-256 із режимом CBC і випадковим IV для шифрування повідомлень захищає від вразливостей кінцевих точок.

Таким чином у рамках кваліфікаційної роботи виконано усі поставлені завдання:

- Проведено огляд популярних месенджерів, досліджено їх безпеку та збереження конфіденційності даних.
- Розглянуто безпеку WhatsApp, зокрема імплементацію протоколу Signal для наскрізного шифрування.
- Проаналізовано атаки, які обходять механізми E2EE, зокрема фішинг та вразливості кінцевих пристроїв.
- Розроблено програмний застосунок, що забезпечує механізм шифрування трафіку для передачі даних через месенджери.

Розроблене програмне рішення ефективно захищає від розглянутих у 2 розділі вразливостей. Модульна архітектура дозволяє легко розширювати систему для підтримки інших месенджерів і алгоритмів шифрування, що підтверджує практичну цінність роботи та її відповідність меті створення безпечного механізму передачі даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Most popular messaging apps 2025| Statista. *Statista*. [Електронний ресурс]. – URL: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/ (дата звернення: 13.01.2025)

2. Рейтинг мобільних додатків за квітень 2022. *Kantar. Shape your brand future.* [Електронний ресурс]. – URL: https://www.kantar.com/ua/inspiration/advertising-media/mobile-app-ranking_april-2022 (дата звернення: 14.01.2025)

3. *Verizon:* Data Breach Investigations Report 2023| [Електронний ресурс]. – https://www.verizon.com/business/resources/Te46/reports/2023-dbir-public-sector-snapshot.pdf (дата звернення: 14.01.2025)

4. The editorial board. Encryption ‘back doors’ are a bad idea. *Financial Times*. [Електронний ресурс]. – https://www.ft.com/content/186a9bb6-9e62-47ce-b974-ce52a2d73e9f (дата звернення: 14.01.2025)

5. Muncaster P. Tech Execs: Multi-Factor Authentication Can Prevent 90% of Attacks. Infosecurity Magazine. [Електронний ресурс]. – URL: https://www.infosecurity-magazine.com/news/tech-execs-mfa-prevent-90-of/ (дата звернення: 20.01.2025)

6. Інформація про наскрізне шифрування | *Довідковий центр WhatsApp*. [Електронний ресурс]. – URL: https://faq.whatsapp.com/820124435853543/?locale=uk_UA (дата звернення: 20.01.2025)

7. BBC News. WhatsApp issued second-largest GDPR fine of €225m. [Електронний ресурс]. – URL: https://www.bbc.com/news/technology-58422465 (дата звернення: 20.01.2025)

8. Data Protection Commission. *Data Protection Commission*. [Електронний ресурс]. – URL: www.dataprotection.ie/en/news-media/data-protection-commission-announces-conclusion-inquiry-whatsapp (дата звернення: 20.01.2025)

9. Documentation. *Signal Messenger* [Електронний ресурс]. – URL: https ://signal.org/docs/ (дата звернення: 25.01.2025)

10. Gerken T. What is messaging app Signal and how secure is it? BBC. [Електронний ресурс]. – URL: https://www.bbc.com/news/articles/c1kjd0910 19o (дата звернення: 25.01.2025)

11. Grand jury subpoena for Signal user data, Eastern District of Virginia. *Signal Messenger*. [Електронний ресурс]. – URL: https://signal.org/bigbrot her/eastern-virginia-grand-jury/ (дата звернення: 25.01.2025)

12. Jakobsen J., Orlandi C. On the CCA (in)Security of MTProto. *CCS'16: 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna Austria. New York, NY, USA, 2016. URL: https://doi.org/10.1145/2994459.2994468

13. Salganik T. Telegram cooperates with Roskomnadzor and the FSB and stores user data indefinitely - Spravdi | УНН. *Ukrainian National News (UNN)*. [Електронний ресурс]. – URL: https://unn.ua/en/news/telegram-cooperates-with-roskomnadzor-and-the-fsb-and-stores-user-data-indefinitely-spravdi (дата звернення: 25.01.2025)

14. Security Viber. *Viber*. [Електронний ресурс]. – URL: https://www.vibe r.com/en/security/ (дата звернення: 27.01.2025)

15. Terms & Policies |Viber. *Viber*. [Електронний ресурс]. – URL: https:// www.viber.com/en/terms/ (дата звернення: 27.01.2025)

16. Greenberg A. Facebook Says Encrypting Messenger by Default Will Take Years. *WIRED*. [Електронний ресурс]. – URL: https://www.wired.com/story/facebo ok-messenger-end-to-end-encryption-default/ (дата звернення: 27.01.2025)

17. Наскрізне шифрування (E2EE): детальний огляд. *IT Education Center Blog*. [Електронний ресурс]. – URL: https://itedu.center/ua/blog/review/naskrizne-shyfruvannia-prostymy-slovamy-pro-e2ee/?srsltid=AfmBOor0LOSQJVrph39s-jl2GrxaUsQ1TznF0rSNp8lgA1iStSsEs86F (дата звернення: 5.02.2025)

18. Julianto A. I., Rimbawa H. A. D., Asnar Y. D. W. Study and Analysis of End-to-End Encryption Message Security Using Diffie-Hellman Key Exchange

Encryption. *International Journal of Progressive Sciences and Technologies*. 2023. Vol. 42, no. 1. P. 173. URL: https://doi.org/10.52155/ijpsat.v42.1.5844

19. Lutkevich B., Bacon M. What is End-to-End Encryption (E2EE) and How Does it Work?. *Search Security*. [Електронний ресурс]. – URL: https://www.techtarget.com/searchsecurity/definition/end-to-end-encryption-E2EE (дата звернення: 13.02.2025)

20. Matthias M. Signal | Overview, History, & Facts | Britannica. *Encyclopedia Britannica*. [Електронний ресурс]. – URL: https://www.britannica.com/topic/Signal-app (дата звернення: 13.02.2025)

21. Specifications >> The X3DH Key Agreement Protocol. Signal Messenger. [Електронний ресурс]. – URL: https://signal.org/docs/specifications/x3dh/ (дата звернення: 24.02.2025)

22. Nidhi Rastogi, James Hendler Rensselaer | WhatsApp security and role of metadata in preserving privacy. *Polytechnic Institute, Troy, NY, USA. URL:* https://arxiv.org/pdf/1701.06817

23. Резервна копія, захищена наскрізним шифруванням | Довідковий центр WhatsApp. [Електронний ресурс]. – URL: https://faq.whatsapp.com/490592613091019 (дата звернення: 24.02.2025)

24. Castro C. WhatsApp encryption isn't the problem, metadata is. *TechRadar*. [Електронний ресурс]. – URL: https://www.techradar.com/computing/cyber-security/whatsapp-encryption-isnt-the-problem-metadata-is (дата звернення: 07.03.2025)

25. Guardian staff reporter. WhatsApp design feature means some encrypted messages could be read by third party. *The Guardian*. [Електронний ресурс]. – URL : https://www.theguardian.com/technology/2017/jan/13/whatsapp-design-feature-encrypted-messages (дата звернення: 07.03.2025)

26. Data and file storage overview | App data and files | Android Developers. *Android Developers*. [Електронний ресурс]. – URL: https://developer.android.com/training/data-storage (дата звернення: 18.03.2025)

27. Symantec Mobile Threat: Attackers Can Manipulate Your WhatsApp and Telegram Media Files. *Symantec Enterprise Blogs*. [Електронний ресурс]. – URL: https://www.security.com/expert-perspectives/symantec-mobile-threat-defense-attackers-can-manipulate-your-whatsapp-and-telegram-media (дата звернення: 18.03.2025)

28. Group G. T. I. Signals of Trouble: Multiple Russia-Aligned Threat Actors Actively Targeting Signal Messenger | Google Cloud Blog. *Google Cloud Blog*. [Електронний ресурс]. – URL: https://cloud.google.com/blog/topics/threat-intelligence/russia-targeting-signal-messenger (дата звернення: 02.04.2025)

29. CERT-UA. Викрадення акаунту WhatsApp під виглядом голосування за електронні петиції (CERTUA#9565). [Електронний ресурс]. – URL: https://cert.gov.ua/article/6278735 (дата звернення: 02.04.2025)

30. Selenium Overview. *Selenium*. [Електронний ресурс]. – URL: https://www.selenium.dev/documentation/overview/ (дата звернення: 02.04.2025)

ДОДАТКИ

Додаток А

Лістинг коду графічного інтерфейсу – MainWindow.xaml

```
<Window x:Class="WhatsApp_2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WhatsApp_2"
        mc:Ignorable="d"
        Title="Secure Chat"
        Height="600"
        Width="575"
        MinWidth="575"
        MaxWidth="575"
        Background="#FFF4F6F8"
        FontSize="14"
        WindowStartupLocation="CenterScreen"
        Icon="icon/lock.ico">
    <Grid Margin="20">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>

        <!-- Контакт -->
        <StackPanel Grid.Row="0" Orientation="Horizontal" Margin="0,0,0,10"
VerticalAlignment="Center" HorizontalAlignment="Left">
            <TextBlock Text="Контакт:" FontWeight="Bold" VerticalAlignment="Center"/>
            <TextBox x:Name="ContactNameTextBox" Width="150" Margin="27,0,0,0" Padding="5"
/>
            <Button Content="Відкрити чат" FontWeight="Bold" Click="OpenChatButton_Click"
Width="110" Margin="9,0" Padding="5" FontSize="14"/>
            <TextBlock x:Name="ContactStatusTextBlock" Text="Контакт не обрано"
FontWeight="Bold" Foreground="Red" VerticalAlignment="Center" Margin="10,0,0,0"/>
        </StackPanel>

        <!-- Пароль -->
     <StackPanel Grid.Row="1" Orientation="Horizontal" Margin="0,0,0,10"
VerticalAlignment="Center" HorizontalAlignment="Left">
            <TextBlock Text="Пароль:" FontWeight="Bold" VerticalAlignment="Center"/>
            <PasswordBox x:Name="PasswordBox" Width="150" Margin="29,0,0,0" Padding="5"/>
            <Button Content="Ввести" FontWeight="Bold" Click="PasswordButton_Click"
Width="110" Margin="10,0" Padding="5"/>
```

Продовження Додатку А

```
        <TextBlock x:Name="PasswordStatusTextBlock" Text="Пароль не введено" FontWeight="Bold" Foreground="Red" VerticalAlignment="Center" Margin="10,0,0,0"/>
    </StackPanel>

    <!-- Кнопка читання -->
    <Button Grid.Row="2" Content="Прочитати чат" Click="ReadButton_Click"
        Height="35" Width="150" HorizontalAlignment="Left"
        Background="#35ab47" Foreground="White" FontWeight="Bold"
        Margin="0,0,0,10" BorderThickness="0"/>
    <Button Grid.Row="2" Content="Надіслати файл" Click="SendFileButton_Click"
        Height="35" Width="150" HorizontalAlignment="Center"
        Background="#35ab47" Foreground="White" FontWeight="Bold"
        Margin="0,0,0,10" BorderThickness="0"/>
    <Button Grid.Row="2" Content="Розшифрувати файл" Click="DecryptFileButton_Click"
        Height="35" Width="150" HorizontalAlignment="Right"
        Background="#35ab47" Foreground="White" FontWeight="Bold"
        Margin="0,0,0,10" BorderThickness="0"/>

    <!-- Повідомлення -->
    <ScrollViewer Grid.Row="3" VerticalScrollBarVisibility="Auto" Margin="0,0,0,10">
      <ListBox x:Name="MessagesListBox" Background="#FFF4F6F8" BorderThickness="0">
        <ListBox.ItemContainerStyle>
          <Style TargetType="ListBoxItem">
            <Setter Property="HorizontalContentAlignment" Value="Stretch"/>
            <Setter Property="Padding" Value="0"/>
            <Setter Property="Margin" Value="0"/>
            <Setter Property="Template">
              <Setter.Value>
                <ControlTemplate TargetType="ListBoxItem">
                  <ContentPresenter HorizontalAlignment="{Binding Alignment}"/>
                </ControlTemplate>
              </Setter.Value>
            </Setter>
          </Style>
        </ListBox.ItemContainerStyle>
        <ListBox.ItemTemplate>
          <DataTemplate>
            <Border Background="{Binding BubbleColor}" CornerRadius="12" Padding="10" Margin="5" MaxWidth="300" >
              <StackPanel>
                <TextBlock Text="{Binding Text}" TextWrapping="Wrap" FontSize="16" Foreground="Black"/>
                <TextBlock Text="{Binding Time}" FontSize="12" Foreground="Gray" HorizontalAlignment="Right" Margin="0,5,0,0"/>
              </StackPanel>
            </Border>
          </DataTemplate>
        </ListBox.ItemTemplate>
      </ListBox>
    </ScrollViewer>
```

Продовження Додатку А

```
        <!-- Ввід повідомлення -->
        <StackPanel Grid.Row="4" Orientation="Horizontal" Margin="0,0,0,0"
VerticalAlignment="Center" >
            <TextBox x:Name="SendTextBox" Width="330" Height="40" Margin="0,0,10,0"
TextWrapping="Wrap"
                     VerticalScrollBarVisibility="Auto" HorizontalAlignment="Left" Padding="5"
FontSize="14"/>
            <Button Content="Надіслати" Click="SendButton_Click"
                    Height="40" Width="170" FontSize="14" HorizontalAlignment="Right"
                    Background="#35ab47" Foreground="White" FontWeight="Bold"
                    BorderThickness="0" />
        </StackPanel>
    </Grid>
</Window>
```

Додаток Б

Лістинг коду UI логіки – MainWindow.xaml.cs

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using Microsoft.Win32;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;
using WhatsApp_2.Interfaces;

namespace WhatsApp_2
{
   public partial class MainWindow : Window
   {
      private IWebDriver _driver;
      private WebDriverWait _wait;
      private IEncryption _encryption;
      private IMessenger _messenger;

      public MainWindow()
      {
         InitializeComponent();
         _driver = new ChromeDriver();
         _driver.Navigate().GoToUrl("https://web.whatsapp.com");
         _wait = new WebDriverWait(_driver, TimeSpan.FromSeconds(10));
         _encryption = new AesEncryption();
         _messenger = new WhatsAppMessenger(_driver, _wait, _encryption);

         MessageBox.Show("Відскануйте QR-код для входу в WhatsApp");
      }

      // Chat Name
      private void OpenChatButton_Click(object sender, RoutedEventArgs e)
      {
         string contactName = ContactNameTextBox.Text;
         try
         {
            _messenger.OpenChat(contactName);
            ContactStatusTextBlock.Text = $"Контакт: {contactName}";
            ContactStatusTextBlock.Foreground = Brushes.Green;

            MessageBox.Show("Чат Відкрито");
         }

         catch (Exception ex)
```

Продовження Додатку Б

```
    {
        MessageBox.Show("Контакт не знайдено");
    }
}

// Password
private void PasswordButton_Click(object sender, RoutedEventArgs e)
{
    string password = PasswordBox.Password;
    _encryption.GenerateKeys(password);
    PasswordStatusTextBlock.Text = "Пароль введено";
    PasswordStatusTextBlock.Foreground = Brushes.Green;
    MessageBox.Show("Пароль введено");
}

// Send message
private void SendButton_Click(object sender, RoutedEventArgs e)
{
    string message = SendTextBox.Text;
    _messenger?.SendMessage(message);
    SendTextBox.Clear();
}

private void ReadButton_Click(object sender, RoutedEventArgs e)
{
    StartMessageRefreshLoop();
}

private CancellationTokenSource _cts;

public class MessageItem
{
    public string Text { get; set; }
    public string Time { get; set; }
    public HorizontalAlignment Alignment { get; set; }
    public Brush BubbleColor { get; set; }
}

private async void StartMessageRefreshLoop()
{
    _cts = new CancellationTokenSource();
    try
    {
        while (!_cts.Token.IsCancellationRequested)
        {
            if (_messenger != null)
            {
                var messageItems = await _messenger.GetMessagesAsync();
                MessagesListBox.Dispatcher.Invoke(() =>
```

Продовження Додатку Б

```
                {
                    MessagesListBox.Items.Clear();
                    foreach (var item in messageItems)
                    {
                        MessagesListBox.Items.Add(item);
                    }
                });
            }
            await Task.Delay(TimeSpan.FromSeconds(5), _cts.Token);
        }
    }
    catch (TaskCanceledException)
    {
        // Цикл зупиняється при закритті прогами
        // методом StopMessageRefreshLoop(), що викликається у OnClosed
    }
}
private void SendFileButton_Click(Object sender, RoutedEventArgs e)
    {
        var dialog = new OpenFileDialog();
        if (dialog.ShowDialog() == true)
        {
            string inputPath = dialog.FileName;
            try
            {
                _messenger.SendEncryptedFile(inputPath);
                SendTextBox.Clear();
                MessageBox.Show("Файл зашифровано і надіслано.");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка відправлення файлу: " + ex.Message);
            }
        }
    }

    private void DecryptFileButton_Click(object sender, RoutedEventArgs e)
    {
        var dialog = new OpenFileDialog();
        if (dialog.ShowDialog() == true)
        {
            string encryptedPath = dialog.FileName;
            try
            {
                string decryptedPath = _messenger.SaveDecryptedFile(encryptedPath);
                MessageBox.Show("Файл розшифровано:\n" + decryptedPath);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка розшифрування: " + ex.Message);
```

Продовження Додатку Б

```
            }
        }
    }

private void StopMessageRefreshLoop()
    {
        _cts?.Cancel();
    }

    protected override void OnClosed(EventArgs e)
    {
        StopMessageRefreshLoop();
        _driver?.Quit();
        base.OnClosed(e);
    }
```

Додаток В

Лістинг коду інтерфейсу IMessenger – IMessenger.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using static WhatsApp_2.MainWindow;

namespace WhatsApp_2.Interfaces
{
   public interface IMessenger
   {
      void OpenChat(string contactName);
      void SendMessage(string message);
      Task<string> ReadMessages();
      Task<IEnumerable<MessageItem>> GetMessagesAsync();
      public void SendEncryptedFile(string originalFilePath);
      public string SaveDecryptedFile(string encryptedPath);
   }
}
```

Додаток Г

Лістинг коду інтерфейсу IEncryption – IEncryption.cs

```
namespace WhatsApp_2.Interfaces
{
    public interface IEncryption
    {
        byte[] Encrypt(string plaintext);
        string Decrypt(byte[] ciphertext);
        void GenerateKeys(string password);
        public byte[] EncryptFile(string filePath);
        public void DecryptFile(byte[] encryptedData, string outputFilePath);
    }
}
```

Додаток Д

Лістинг коду взаємодії з месенджером – MessengerService.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using OpenQA.Selenium;
using OpenQA.Selenium.Support.UI;
using System.Threading;
using WhatsApp_2.Interfaces;
using System.Windows;
using System.Collections.Specialized;
using static WhatsApp_2.MainWindow;
using System.Text.RegularExpressions;
using System.Windows.Media;

namespace WhatsApp_2
{

   public class WhatsAppMessenger : IMessenger
   {
      private readonly IWebDriver _driver;
      private readonly WebDriverWait _wait;
      private readonly IEncryption _encryption;

      public WhatsAppMessenger(IWebDriver driver, WebDriverWait wait, IEncryption encryption)
      {
         _driver = driver;
         _wait = wait;
         _encryption = encryption;
      }

      public void OpenChat(string contactName)
      {
         var searchBox = _wait.Until(d => d.FindElement(
            By.XPath("//div[@contenteditable='true']")));
         searchBox.SendKeys(Keys.Control + "a");
         searchBox.SendKeys(Keys.Delete);
         searchBox.SendKeys(contactName);
         Thread.Sleep(2000);
         var contact = _wait.Until(d => d.FindElement(
            By.XPath($"//span[@title='{contactName}']")));
         contact.Click();
      }

      public void SendMessage(string message)
      {

         var encryptedMessage = _encryption.Encrypt(message);
```

Продовження Додатку Д

```
        string base64Message = Convert.ToBase64String(encryptedMessage);

        var messageBox = _wait.Until(d => d.FindElement(By.CssSelector("div[contenteditable='true']" +
            "[data-tab='10']")));
        messageBox.SendKeys(base64Message);
        messageBox.SendKeys(Keys.Enter);
    }

    public async Task<string> ReadMessages()
    {
        var messages = _driver.FindElements(By.CssSelector("span.selectable-text"));
        StringBuilder sb = new StringBuilder();
        foreach (var message in messages)
        {
            var parentElement = message.FindElement(By.XPath("..")).FindElement(By.XPath(".."));
            string sender = parentElement.GetAttribute("data-pre-plain-text");
            string messageText = message.Text;
            string output;

            try
            {
                byte[] encryptedBytes = Convert.FromBase64String(messageText);
                string decryptedMessage = _encryption.Decrypt(encryptedBytes);

                if (!string.IsNullOrWhiteSpace(decryptedMessage))
                {
                    sb.AppendLine($"{sender} {decryptedMessage}");
                }
            }
            catch
            {
                continue;
            }
        }
        return sb.ToString();
    }
    public async Task<IEnumerable<MessageItem>> GetMessagesAsync()
    {
        var messageItems = new List<MessageItem>();
        string newMessages = await ReadMessages();
        var lines = newMessages.Split(new[] { "\r\n", "\n" }, StringSplitOptions.RemoveEmptyEntries);

        foreach (var rawLine in lines)
        {
            string timePart = "", sender = "", message = "";
            var match = Regex.Match(rawLine, @"\[(.*?)\]\s+(.*?):\s+(.*)");
```

Продовження Додатку Д

```
            timePart = match.Groups[1].Value;
            sender = match.Groups[2].Value;
            message = match.Groups[3].Value;
            bool isMe = sender.Equals("Ps", StringComparison.OrdinalIgnoreCase);
            messageItems.Add(new MessageItem
            {
               Text = message,
               Time = timePart,
               Alignment = isMe ? HorizontalAlignment.Right : HorizontalAlignment.Left,
               BubbleColor = isMe ? Brushes.LightGreen : Brushes.LightGray
            });
         }

         return messageItems;
      }

      public void SendEncryptedFile(string originalFilePath)
      {
         try
         {
            byte[] encryptedData = _encryption.EncryptFile(originalFilePath);
            string fileName = System.IO.Path.GetFileName(originalFilePath);
            var encFileBytes = _encryption.Encrypt(fileName);
            string encryptedFileName = Convert.ToHexString(encFileBytes);
            string tempPath = System.IO.Path.Combine(System.IO.Path.GetTempPath(),
encryptedFileName);
            File.WriteAllBytes(tempPath, encryptedData);
            StringCollection fileCollection = new StringCollection();
            fileCollection.Add(tempPath);
            Clipboard.SetFileDropList(fileCollection);
            var messageBox = _wait.Until(d =>
d.FindElement(By.CssSelector("div[contenteditable='true']" +
               "[data-tab='10']")));
            messageBox.Click();
            Thread.Sleep(500);
            messageBox.SendKeys(Keys.Control + "v");
            Thread.Sleep(1500);
            var sendButton = _wait.Until(d => d.FindElement(By.CssSelector("div[role='button'] " +
               "span[data-icon='wds-ic-send-filled']"))).FindElement(By.XPath(".."));
// піднімаємося до батьківського <div>, бо клік треба саме на нього
            sendButton.Click();
            string fileFullName = $"{fileName} {encryptedFileName}";
            SendMessage(fileFullName);
         }
         catch (Exception ex)
         {
            throw new InvalidOperationException($"Помилка відправлення файлу: {ex.Message}",
ex);
         }
      }
```

Продовження Додатку Д

```
public string SaveDecryptedFile(string encryptedPath)
      {
         try
         {
            byte[] encryptedData = File.ReadAllBytes(encryptedPath);
            string dir = System.IO.Path.GetDirectoryName(encryptedPath);

            string encryptedfileName = System.IO.Path.GetFileName(encryptedPath);
            string decryptedfileName =
_encryption.Decrypt(Convert.FromHexString(encryptedfileName));

            string decryptedPath = System.IO.Path.Combine(dir, "decrypted_" + decryptedfileName);

            _encryption.DecryptFile(encryptedData, decryptedPath);

            return decryptedPath;
         }
         catch (Exception ex)
         {
            throw new InvalidOperationException($"Помилка розшифування файлу:
{ex.Message}", ex);
         }
      }
   }
```

Додаток Е

Лістинг коду реалізації шифрування – AesEncryption.cs

```
using System;
using System.Linq;
using System.Text;
using WhatsApp_2.Interfaces;
using System.IO;
using System.Security.Cryptography;

namespace WhatsApp_2
{
   public class AesEncryption : IEncryption
   {
      private byte[] _key;

      public void GenerateKeys(string password)
      {
         var hash = SHA256.HashData(Encoding.UTF8.GetBytes(password));
         _key = hash;
      }

      public byte[] Encrypt(string plaintext)
      {
         if (plaintext == null || plaintext.Length <= 0)
            throw new ArgumentNullException("plainText");
         if (_key == null || _key.Length <= 0)
            throw new ArgumentNullException("Key");

         using var aesAlg = Aes.Create();
         aesAlg.Key = _key;
         aesAlg.GenerateIV();

         var encryptor = aesAlg.CreateEncryptor();
         using var msEncrypt = new MemoryStream();

         msEncrypt.Write(aesAlg.IV, 0, aesAlg.IV.Length);

         using (var csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
         {
            var plainBytes = Encoding.UTF8.GetBytes(plaintext);
            csEncrypt.Write(plainBytes, 0, plainBytes.Length);
            csEncrypt.FlushFinalBlock();
         }

         return msEncrypt.ToArray(); // [IV | CipherText]
      }

      public string Decrypt(byte[] ciphertext)

      {
```

Продовження Додатку Е

```
    if (ciphertext == null || ciphertext.Length <= 0)
        throw new ArgumentNullException("cipherText");
    if (_key == null || _key.Length <= 0)
        throw new ArgumentNullException("Key");

    using var aesAlg = Aes.Create();
    aesAlg.Key = _key;

    // Виділяємо IV з перших 16 байтів
    var iv = ciphertext.Take(16).ToArray();
    aesAlg.IV = iv;

    var actualCipher = ciphertext.Skip(16).ToArray();
    var decryptor = aesAlg.CreateDecryptor();

    using var msDecrypt = new MemoryStream(actualCipher);
    using var csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read);
    using var msPlain = new MemoryStream();
    csDecrypt.CopyTo(msPlain);
    return Encoding.UTF8.GetString(msPlain.ToArray());
}

public byte[] EncryptFile(string filePath)
{
    if (filePath == null || filePath.Length <= 0)
        throw new ArgumentNullException("filePath");
    if (_key == null || _key.Length <= 0)
        throw new ArgumentNullException("Key");

    using var aesAlg = Aes.Create();
    aesAlg.Key = _key;
    aesAlg.GenerateIV();

    using var msEncrypt = new MemoryStream();
    msEncrypt.Write(aesAlg.IV, 0, aesAlg.IV.Length);

    var encryptor = aesAlg.CreateEncryptor();
    using (var csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
    using (var fsInput = File.OpenRead(filePath))
    {
        fsInput.CopyTo(csEncrypt);
        csEncrypt.FlushFinalBlock();
    }

    return msEncrypt.ToArray();
}

public void DecryptFile(byte[] encryptedData, string outputFilePath)
{
```

Продовження Додатку Е

```
        if (outputFilePath == null || outputFilePath.Length <= 0)
            throw new ArgumentNullException("outputFilePath");
        if (_key == null || _key.Length <= 0)
            throw new ArgumentNullException("Key");

        using var aesAlg = Aes.Create();
        aesAlg.Key = _key;

        var iv = encryptedData.Take(16).ToArray();
        aesAlg.IV = iv;

        var actualCipher = encryptedData.Skip(16).ToArray();
        var decryptor = aesAlg.CreateDecryptor();

        using var msDecrypt = new MemoryStream(actualCipher);
        using var csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read);
        using var fsOutput = File.Create(outputFilePath);
        csDecrypt.CopyTo(fsOutput);
    }
}
```