МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

ПЕРЕРВА Дмитро Ігорович

Алгоритми шифрування для підвищення безпеки обміну повідомленнями / Encryption Algorithms for Enhancing the Security of Message Exchange

спеціальність: 125 – Кібербезпека та захист інформації

освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи
КБм-21
Д.І. Перерва

Науковий керівник
В.В. Яцків

Кваліфікаційну роботу
Допущено до захисту:

«____» _____ ___2025 р.

Завідувач кафедри
_____ **В.В.Яцків**

ТЕРНОПІЛЬ - 2025

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.В.Яцків
«____» _____ 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ПЕРЕРВА Дмитро Ігорович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:

Алгоритми шифрування для підвищення безпеки обміну повідомленнями / Encryption Algorithms for Enhancing the Security of Message Exchange

керівник роботи д.т.н. В.В. Яцків

затверджені наказом по університету від 20 грудня 2024 року № 938

2. Строк подання студентом закінченої кваліфікаційної роботи 5 грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити:

– провести аналіз сучасних протоколів захищеного обміну повідомленнями та оцінити їх уразливість до витоку метаданих.

– дослідити криптографічні основи симетричного шифрування та обґрунтувати вибір AES-GCM як базового алгоритму.

– проаналізувати архітектуру та механізми роботи месенджера Session у контексті формування та передачі метаданих.

– розробити модифікований алгоритм шифрування для зменшення витоку метаданих у системах безпечного обміну повідомленнями.

– реалізувати алгоритм програмно та здійснити його тестування на реальних сценаріях обміну повідомленнями.

– виконати порівняльну оцінку ефективності та стійкості розробленого підходу відносно стандартної реалізації AES-GCM та механізмів Session.

5. Перелік графічного матеріалу у роботі:

– принцип роботи E2E(наскрізного шифрування);

– схематичне порівняння: симетрія / асиметрія;

– приклад структури блочного шифрування;

– Спрощена схема HKDF-Extract/Expand.

6. Консультанти розділів кваліфікаційної роботи

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз сучасних алгоритмів шифрування та їх використання у системах обміну повідомленнями	12.2024 р. – 03.2025 р.	
2	Сучасні протоколи та підходи до забезпечення безпечного обміну даними	03.2025 р. – 06.2025 р.	
3	Реалізація алгоритмів шифрування на базі месенджера "Session"	06.2025 р. – 11.2025 р.	

Студент _____ Перерва Д.І.
(підпис)

Керівник роботи _____ д.т.н. В.В. Яцків
(підпис)

АНОТАЦІЯ

Перерва Д. І. Алгоритми шифрування для підвищення безпеки обміну повідомленнями. – Рукопис.

Дослідження на здобуття освітнього ступеня «магістр» за спеціальністю 125 «Кібербезпека та захист інформації», освітньо-професійна програма «Кібербезпека». – Західноукраїнський національний університет, Тернопіль, 2025.

У роботі проведено аналіз сучасних протоколів захищеного обміну повідомленнями та визначено ключові проблеми, пов’язані з витоком метаданих. Досліджено архітектуру та принципи функціонування месенджера Session, зокрема механізми формування службової інформації під час передачі повідомлень. Обґрунтовано вибір алгоритму AES-GCM як базового інструмента шифрування та запропоновано його модифіковану схему для зменшення обсягів метаданих. Реалізовано програмну модель алгоритму, проведено тестування та оцінено ефективність порівняно зі стандартною реалізацією.

Ключові слова: БЕЗПЕЧНИЙ ОБМІН ПОВІДОМЛЕННЯМИ, AES-GCM, КРИПТОАЛГОРИТМИ, СИСТЕМИ ПОВІДОМЛЕНЬ.

ABSTRACT

Pererva D. I. Encryption Algorithms for Enhancing the Security of Message Exchange. – Manuscript.

Research submitted for the Master's degree in specialty 125 "Cybersecurity and Information Protection", Educational and Professional Program "Cybersecurity". – West Ukrainian National University, Ternopil, 2025.

The thesis presents an analysis of modern secure messaging protocols and identifies the key challenges related to metadata leakage. The architecture and operational principles of the Session messenger are examined, with a focus on mechanisms that generate service information during message transmission. The selection of the AES-GCM encryption algorithm as the core cryptographic primitive is theoretically justified, and a modified scheme aimed at reducing metadata exposure is proposed. A software implementation of the improved algorithm has been developed, followed by comprehensive testing and performance evaluation in comparison with the standard AES-GCM workflow.

Keywords: SECURE MESSAGING, AES-GCM, CRYPTOGRAPHIC ALGORITHMS, MESSAGING SYSTEMS.

ЗМІСТ

ВСТУП

Актуальність роботи. Сучасні системи електронної комунікації характеризуються значним зростанням обсягів передавання даних та активним використанням мобільних застосунків для обміну повідомленнями. Ці процеси супроводжуються підвищенням вимог до захисту інформації, зокрема до забезпечення конфіденційності, цілісності та приватності користувачів. Навіть за умови застосування класичних алгоритмів шифрування, суттєвою загрозою залишається можливість витоку метаданих, аналізу структури трафіку та побудови соціальних графів взаємодії. Це обумовлює потребу у вдосконаленні криптографічних механізмів, здатних гарантувати безпеку як змісту, так і службової інформації [1–4].

Проблематика забезпечення захищеного обміну повідомленнями особливо актуальна у контексті використання децентралізованих та анонімних месенджерів, де розподілена архітектура передбачає відсутність єдиного центру довіри та опору на незалежні вузли. Для таких систем застосовуються гібридні криптографічні рішення, що включають симетричні алгоритми (AES-GCM, ChaCha20), асиметричні механізми на основі еліптичних кривих (X25519), а також протоколи багаторазової ротації ключів (Double Ratchet). Однак за умови зростання обчислювальних можливостей та складності атак, зокрема аналізу трафіку та кореляційних методів, постає питання удосконалення існуючих схем шифрування для підвищення стійкості до таких загроз.

Особливе значення має проблема мінімізації метаданих, оскільки саме вони часто є джерелом інформації для атакувальників навіть при повністю зашифрованому змісті повідомлень. Месенджери нового покоління, зокрема «Session», реалізують підхід до побудови приватних комунікацій без використання телефонних номерів, IP-ідентифікаторів або централізованих серверів, що створює нові можливості для дослідження та вдосконалення алгоритмів шифрування на рівні прикладної реалізації.

Мета і завдання дослідження. Метою роботи є аналіз, удосконалення та дослідження алгоритмів шифрування, що застосовуються для захищеного обміну повідомленнями, а також їх інтеграція у децентралізовану архітектуру месенджера «Session» з метою підвищення стійкості до аналізу трафіку та витоку метаданих.

Досягнення поставленої мети передбачає розв'язання таких завдань:

– провести аналіз сучасних алгоритмів симетричного та асиметричного шифрування, оцінити їх переваги та недоліки в системах обміну повідомленнями;

– дослідити протоколи end-to-end encryption та механізми обміну ключами у децентралізованих комунікаційних системах;

– розглянути архітектуру та криптографічні механізми, реалізовані у месенджері «Session», і визначити критичні місця, що призводять до витоку метаданих;

– обґрунтувати необхідність удосконалення існуючого алгоритму шифрування на базі AES-GCM;

– розробити модифікований алгоритм симетричного шифрування з використанням додаткових компонентів ключового матеріалу та механізмів рандомізації;

– виконати програмну реалізацію запропонованого рішення і протестувати його у відтворюваному середовищі;

– провести порівняльний аналіз з вихідною реалізацією щодо продуктивності, стійкості до пасивного аналізу та ризику витоку службових ознак;

– дослідити перспективи застосування модифікованих криптографічних алгоритмів для посилення приватності в реальних безпечних месенджерах.

Об'єкт дослідження – процеси забезпечення захищеного обміну повідомленнями в мережевих комунікаційних системах.

Предмет дослідження – алгоритми шифрування та протоколи передачі даних, що використовуються для захисту повідомлень і метаданих у сучасних месенджерах.

Методи дослідження. Під час проведення дослідження використано: математичний аналіз криптографічних алгоритмів, методи симетричного та асиметричного шифрування, теорію побудови протоколів E2EE, експериментальне моделювання, аналіз продуктивності, методи порівняльного тестування, програмну інженерію та моделювання мережевих сценаріїв.

Наукова новизна одержаних результатів. У роботі удосконалено алгоритм симетричного шифрування на основі AES-GCM, що застосовується у протоколах месенджера «Session». Запропоновано використання додаткових компонентів ключового матеріалу та розширеного формування AAD-структур, що сприяє зменшенню витоку метаданих та підвищенню стійкості комунікацій до аналізу трафіку.

Практичне значення отриманих результатів. Розроблено та впроваджено програмну реалізацію модифікованого алгоритму шифрування, адаптовану для інтеграції в архітектуру «Session». Проведено експериментальне дослідження швидкодії та стійкості рішення, що дозволяє рекомендувати його для використання у системах, орієнтованих на підвищений рівень приватності.

Публікації та апробація КР.

Перерва Д. Алгоритми шифрування для підвищення безпеки обміну повідомленнями. Матеріали науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ - 2025), Тернопіль, 2025. – С. 108-110.

Перерва Д. Удосконалені підходи до зменшення витоку метаданих у системах безпечного обміну повідомленнями. Матеріали науково-практичного симпозіуму «Захист інформації», Тернопіль, 2025. – С. 62-64.

РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ АЛГОРИТМІВ ШИФРУВАННЯ ТА ЇХ ВИКОРИСТАННЯ У СИСТЕМАХ ОБМІНУ ПОВІДОМЛЕННЯМИ

1.1. Роль криптографії у захисті інформації під час передавання повідомлень

У сучасних умовах обмін інформацією в мережах став звичайною частиною повсякденного життя. Передача текстових повідомлень, файлів, мультимедійних даних та інших цифрових елементів між користувачами проходить через різні канали зв’язку, які не є захищеними від стороннього доступу. Основною проблемою є те, що дані можуть перехоплюватись на різних ділянках траси: Wi-Fi, мобільні мережі, маршрутизатори, сервери провайдерів, кешуючі вузли. У багатьох випадках користувач не має контролю над тим, як і де саме інформація обробляється. Саме тому виникає потреба застосовувати шифрування для підвищення безпеки обміну повідомленнями [33].

Шифрування дозволяє перетворювати звичайний текст у форму, яка не читається сторонньою особою. Розшифрувати повідомлення може тільки той, хто має правильний ключ. Відповідно, без ключа вміст повідомлення стає практично недоступним для аналізу. Сам факт передачі даних при цьому не зникає, але їх зміст захищений. Це важливо, тому що самі канали зв’язку не можна вважати надійними навіть при використанні авторизованих мереж.

Необхідність використання шифрування також зумовлена тим, що сьогодні кіберзагрози зростають. Поширені сценарії перехоплення даних включають активні атаки, пасивний аналіз трафіку, підміну ключів, атаки “людина посередині”, логування даних на проміжних серверах. Якщо система не використовує механізми криптографічного захисту, то будь-який учасник маршрутизації пакету може отримати доступ до вмісту повідомлення. Це може відбуватися навіть без явного злому [34].

Варто відзначити, що більшість сучасних популярних месенджерів вже перейшли на шифрування кінцевої точки. Це означає, що повідомлення

шифрується на пристрої відправника і розшифровується тільки на пристрої отримувача. Навіть сервер, через який проходить повідомлення, не може прочитати його зміст. Така модель значно підвищує рівень конфіденційності у порівнянні з класичними варіантами, де шифрування виконувалось тільки між клієнтом та сервером [1, 24].

Слід також зазначити, що обмін повідомленнями може мати різні моделі роботи. У деяких випадках дані передаються у режимі реального часу, у інших – сервер виконує функцію проміжного сховища, де повідомлення зберігаються до моменту доставки. Це не змінює самої логіки небезпеки. Якщо повідомлення зберігаються на сервері у відкритому вигляді, вони можуть потрапити до сторонніх осіб у разі витоку або внутрішнього доступу персоналу.

Тому загальним підходом є використання гібридних криптографічних структур. Асиметрія вирішує питання передачі ключів, а симетрія дозволяє виконувати швидке шифрування без підвищених обчислювальних витрат. Така модель застосовується і у web-комунікаціях, і у месенджерах, і у корпоративних рішеннях (рисунок 1.1).

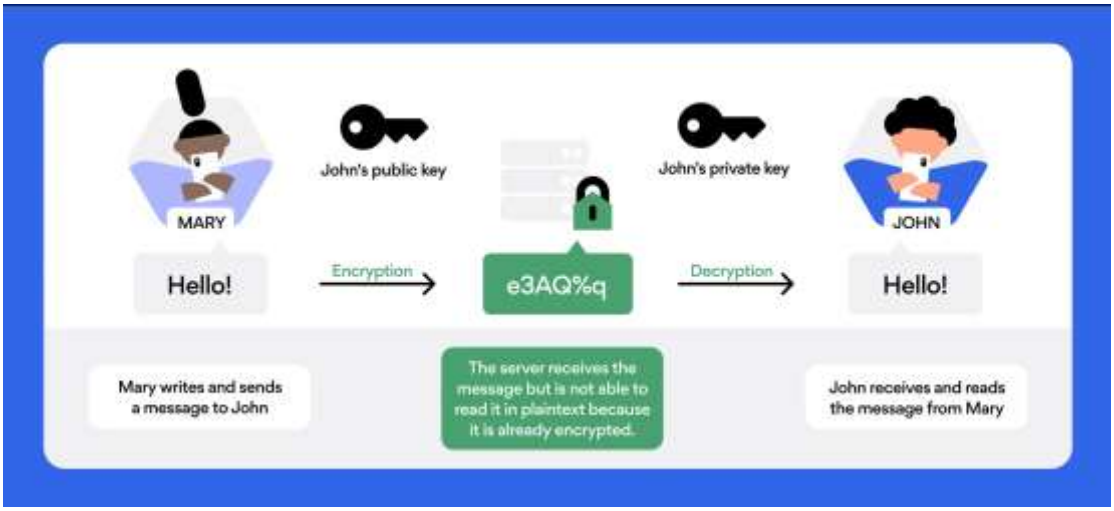


Рисунок 1.1 – Принцип роботи E2E (наскрізного шифрування) [37]

Коли говорять про безпеку в контексті передавання даних, більшість людей уявляє собі ситуацію, коли хтось “ламає” сервіс або отримує доступ до чужого акаунта. Але в мережевій безпеці не менш важливим є момент

перехоплення самого каналу зв’язку. Передача повідомлення – це завжди переміщення даних через проміжні вузли. Саме там і можливі більшість ризиків. Якщо не використовується шифрування, то фактично повідомлення є тим самим, що і листівка, яку всі по дорозі можуть переглянути або сфотографувати [36].

У мережах загального користування це особливо відчувається. Якщо взяти, наприклад, публічні точки доступу Wi-Fi, які є у кафе, аеропортах, вокзалах, торгівельних центрах, то там трафік проходить через загальний доступ. Тому без шифрування повідомлення схопити не складає проблеми. Навіть якщо користувач не підозрює, що по каналу йде перехоплення, це не означає, що його немає. Такі атаки можуть виконуватись пасивно, без створення помітних ознак втручання. І саме тому шифрування введено у стандарти, як елемент базового рівня безпеки.

Ще один аспект, про який часто забувають – це те, що криптографія вирішує не тільки проблему “щоб ніхто не прочитав”. Також важливо гарантувати, що повідомлення не було змінене. Це називається цілісність даних. Якщо дані шифруються належним чином, і шифрування забезпечує тег автентичності, то будь-яка зміна в зашифрованому повідомленні робить його недійсним. У такій ситуації атакувальник не може “вклеїти” своє повідомлення замість справжнього, і тому можливість підміни блокується.

Крім технічних причин, є і питання довіри. Користувачі звикли до певного рівня приватності у цифровому спілкуванні. І коли месенджери перестали бути просто додатками для текстових повідомлень, а стали використовуватись для передавання робочих файлів, особистих зображень, конфіденційних документів, важливість шифрування зросла. Тому провайдери зрозуміли, що без end-to-end шифрування їхні продукти не будуть конкурентними.

Шифрування стало тим механізмом, через який користувач може бути впевнений, що повідомлення бачитиме тільки отримувач. І на практиці, це є основою захищеного обміну даними. Навіть якщо сервер не розуміє вмісту

повідомлень, цього достатньо щоб зняти з нього відповідальність щодо приватності. Контроль над змістом повертається до кінцевих сторін – відправника і отримувача [27].

(Тут можна вставити рисунок – загальна логіка E2EE: відправник → шифрування → мережа → розшифрування → отримувач)

Важливо відмітити, що без криптографії неможливо забезпечити конфіденційність обміну повідомленнями не тільки у відкритих мережах. Навіть у ситуаціях, де передача йде через інфраструктуру оператора або у закритій корпоративній мережі, ризики нікуди не зникають. Будь-яка точка доступу, вузол маршрутизації, логування на проміжних серверах або хмарних платформах може стати місцем витоку. Тому проблема не обмежується тільки “публічним Wi-Fi”. Сьогодні для захисту даних найкращим рішенням є розміщення захисту безпосередньо на рівні користувача, а не на стороні сервера.

Криптографія стала тим інструментом, який дозволяє незалежно від каналу зв’язку гарантувати, що зміст повідомлення не буде відкритим для сторонньої сторони. Таким чином, шифрування перестало бути “додатковою опцією” і стало базовим елементом архітектури сучасних месенджерів. І це сформувало іншу вимогу: необхідність не просто шифрувати, а робити це швидко, ефективно і з мінімальними затримками. Якщо алгоритм складний, але повільний – користувачі від цього тільки програють, навіть якщо його стійкість на висоті.

Тому у подальшому аналізі важливо не лише визначити загальну роль криптографії, а й розібрати які саме алгоритми використовуються, у чому різниця між ними і чому одні базуються на симетричних ключах, а інші – на відкритих ключах. Це є ключовим моментом для розуміння того, як будується захищений обмін повідомленнями на практиці.

Оскільки сучасні системи переважно використовують гібридну побудову, де одні алгоритми є допоміжними для встановлення секретного сеансового ключа, а інші працюють вже безпосередньо з даними, доцільно окремо

розглянути саме класифікацію алгоритмів шифрування. Це дозволить побачити, чому є різні підходи і які саме задачі вони вирішують. Далі у роботі буде наведено огляд симетричних та асиметричних алгоритмів, що використовуються для шифрування повідомлень у месенджерах, та пояснено, які особливості роблять їх придатними до застосування у сучасних системах.

1.2. Огляд симетричних та асиметричних алгоритмів шифрування

Алгоритми шифрування можна поділити на дві основні групи: симетричні та асиметричні. Такий поділ важливий, тому що вони виконують різні задачі у процесі захисту повідомлень. Обидва типи використовуються у сучасних системах, але з різними ролями. Це зумовлено тим, що їх сильні сторони не збігаються. Одні працюють швидко, але потребують того, щоб секретний ключ був уже відомий обом сторонам. Інші дозволяють домовитись про секретний ключ прямо через інтернет, але працюють повільніше. Тому для практичних реалізацій зазвичай застосовують гібридну модель.

Симетричне шифрування – це той випадок, коли один і той самий ключ використовується і для шифрування, і для розшифрування. Це означає, що учасники обміну повідомленнями повинні мати один спільний секрет. Перевага симетричного підходу в тому, що він є дуже швидким. Він не потребує важких математичних операцій. Дані шифруються блочно або потоково, і затримка при передачі практично не відчутна. Саме тому симетричні алгоритми використовуються для шифрування великих обсягів інформації. Найбільш поширеним прикладом симетричного алгоритму є AES. Його використовують у протоколах, які є стандартними практично у всіх захищених системах. Також можна згадати ChaCha20. Він не побудований на блочній схемі як AES, але теж забезпечує швидке шифрування. ChaCha20 часто застосовується там, де важлива швидкість і стабільність при слабких процесорах, наприклад у мобільних пристроях.

Асіметричне шифрування працює інакше. Воно базується на використанні двох ключів – відкритого і приватного. Приватний ключ зберігається у таємниці. Відкритий ключ може бути переданий будь-кому. Принцип у тому, що якщо повідомлення зашифрувати відкритим ключем отримувача, то розшифрувати його зможе тільки той, у кого є відповідний приватний ключ. Це вирішує проблему того, як передати секрет без небезпеки. Наприклад, можна надіслати відкритий ключ, і навіть якщо його хтось перехопить, все одно розшифрувати повідомлення він не зможе. Найвідомішими прикладами таких алгоритмів є RSA та алгоритми, які працюють на основі еліптичних кривих. ECC дає можливість отримати таку ж криптостійкість, але при менших розмірах ключа і меншій кількості обчислень. Це робить ECC привабливим для використання у мобільних пристроях.

Симетричні алгоритми в основному обираються для шифрування самих повідомлень, тому вони працюють швидко. Асиметричні методи призначені переважно для формування або узгодження ключів. Це означає, що вони допомагають створити секрет між двома сторонами. Але фактичне шифрування даних виконується симетричним методом. Тобто у реальних системах зазвичай працює наступна логіка: асиметричний алгоритм використовується для обміну ключем, а симетричний – для шифрування вмісту повідомлень.

Така модель виявилась настільки зручною, що її використовують у всіх популярних месенджерах. Це дозволяє мінімізувати затримки, не знижуючи рівень криптографічного захисту. Тому коли ми говоримо про шифрування повідомлень, варто пам’ятати, що жоден тип алгоритму сам по собі не є достатнім. Їх сила проявляється саме у поєднанні (рисунок 1.2).

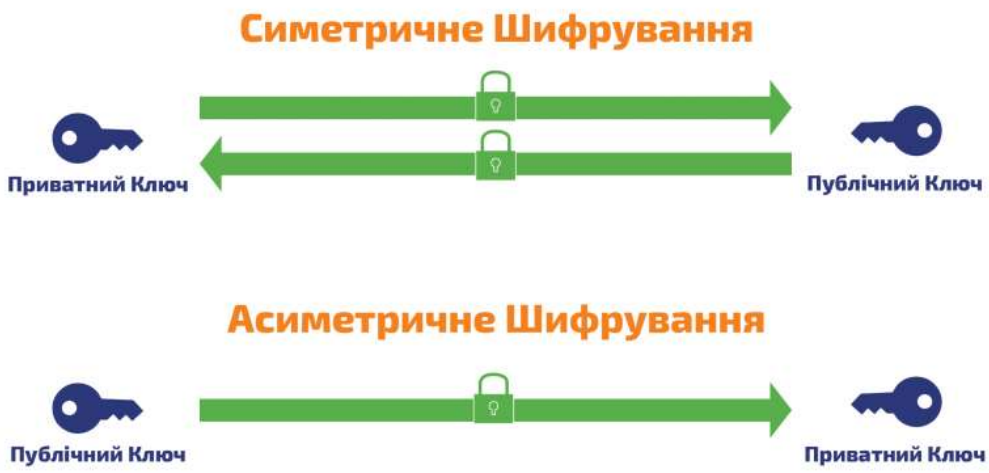


Рисунок 1.2 – Схематичне порівняння: симетрія / асиметрія [38]

Щоб краще зрозуміти особливості алгоритмів, доцільно розглянути кожний з них окремо. Це дозволить звернути увагу на принципи роботи, переваги, обмеження, та причини, чому конкретні алгоритми застосовуються або не застосовуються у практичних системах захищеного обміну повідомленнями. У подальших підрозділах буде наведено характеристику найбільш розповсюджених алгоритмів, починаючи з AES.

Щоб краще зрозуміти особливості алгоритмів, доцільно розглянути кожний з них окремо. Це дозволить звернути увагу на принципи роботи, переваги, обмеження, та причини, чому конкретні алгоритми застосовуються або не застосовуються у практичних системах захищеного обміну повідомленнями. У подальших підрозділах буде наведено характеристику найбільш розповсюджених алгоритмів, починаючи з AES.

1.2.1. Алгоритм AES

AES є одним з найбільш відомих та використовуваних алгоритмів симетричного шифрування. Він був обраний як стандарт заміни застарілого DES. Основною причиною його поширення є поєднання високої швидкості виконання та достатньої криптостійкості. Його використовують у практично всіх сферах: від шифрування даних у мережах, до захисту локальних файлів на комп’ютерах і мобільних телефонах.

У контексті обміну повідомленнями AES застосовується у режимі роботи GCM. Це дозволяє не лише зашифрувати дані, але й перевіряти їх цілісність. GCM працює з так званим тегом автентичності. Якщо дані було змінено, перевірка тега покаже, що повідомлення підроблене. Таким чином AES у режимі GCM фактично виконує дві функції одночасно: шифрує дані та забезпечує автентифікацію [20.

Однією з головних причин, чому AES так активно застосовується у сучасному захисті, є його продуктивність. Він виконує шифрування без створення значного навантаження на процесор. Це важливо саме у месенджерах, де час обробки одного повідомлення має бути мінімальним. AES добре працює навіть на мобільних пристроях. Важливо і те, що AES не прив'язаний до конкретної апаратної архітектури. Це означає, що він може бути реалізований у різних системах, від простих контролерів до повноцінних серверів.

У сучасних системах AES частіше за все використовується не як "сам по собі" алгоритм, а у комбінації з іншими засобами. Він виконує функцію швидкого шифрування даних після того, як ключ вже сформований. Саме тому його рідко обговорюють поза контекстом гібридних систем.

Додатковою перевагою AES є можливість вибору довжини ключа. Існують варіанти з 128, 192 та 256 бітами. Це дозволяє підвищувати криптостійкість у випадку, коли є підозра щодо можливих майбутніх обчислювальних загроз. Наприклад, якщо система орієнтується на застосування у середовищах з підвищеними вимогами до безпеки, часто обирається саме AES-256. У той час як для звичайного побутового комунікаційного обмінення AES-128 також є більш ніж достатнім для сучасних потреб [4, 5].

AES має блочну структуру, де дані розділяються на фрагменти фіксованої довжини, і над цими фрагментами виконується послідовність операцій. У GCM режимі реалізується поєднання потокового принципу та додаткової автентифікаційної перевірки. Важливим елементом у цьому механізмі є IV, який відомий також як вектор ініціалізації [6, 7]. Це випадковий фрагмент даних, що прикріплюється до кожного сеансу шифрування, і завдяки цьому

навіть два однакові повідомлення при використанні одного і того ж ключа матимуть різний зашифрований результат [14, 15].

Це означає, що AES-GCM мінімізує проблему так званого повторного використання ключа. Аналіз трафіку стає значно складнішим для сторонньої особи. Навіть при великому обсязі перехоплених зашифрованих повідомлень немає можливості за знайденими сталими закономірностями однозначно визначити структуру вихідного тексту.

У реальних системах, таких як месенджери, AES не використовується “в чистому вигляді”. Для того, щоб алгоритм працював у конкретному додатку, потрібно виконати організацію формування ключа, його зберігання та передачу. Тому AES зазвичай підключається у момент, коли питання формування ключа вже вирішено іншим механізмом, як правило асиметричним. Тобто AES отримує на вхід уже готовий симетричний ключ і просто виконує шифрування даних.

Таким чином роль AES у контексті захищеного обміну повідомленнями можна описати досить просто: швидко та надійно шифрує зміст. Він є одним з найважливіших елементів у всій системі, але сам по собі не вирішує всіх проблем. Щоб це працювало у повному масштабі, потрібен надійний спосіб узгодження ключа, а це вже інша категорія алгоритмів (рисунок 1.3).

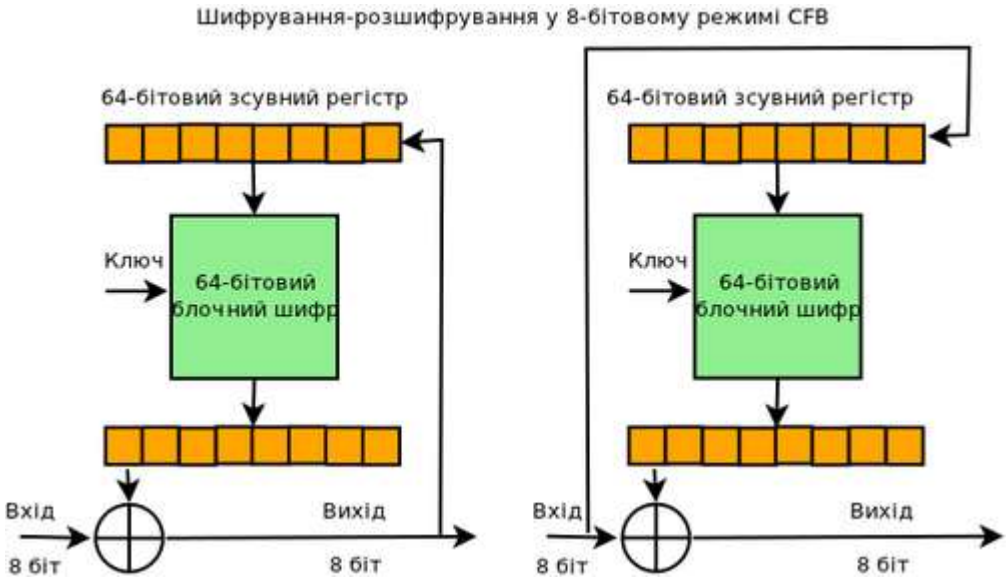


Рисунок 1.3 – Приклад структури блочного шифрування [39]

1.2.2. Алгоритм ChaCha20

ChaCha20 – це сучасний потоковий алгоритм шифрування, який став реальною альтернативою AES у багатьох системах. Його основна особливість у тому, що він побудований не на блочному принципі, а на генеруванні криптографічного потоку байтів, який комбінується з вихідними даними за допомогою операції XOR. Завдяки цьому процес шифрування не залежить від розміру повідомлення і виконується швидко навіть на слабких пристроях.

ChaCha20 розроблено як спрощену і водночас більш стійку модифікацію алгоритму Salsa20. Його творцем є Даніель Бернштейн – той самий криптограф, який створив Curve25519, що використовується у сучасних протоколах обміну ключами. Тому не дивно, що ChaCha20 і Curve25519 часто застосовуються разом у практичних реалізаціях, наприклад у Signal Protocol [13].

Головна перевага ChaCha20 полягає у його стабільній швидкодії на пристроях, де відсутні апаратні прискорювачі AES. У багатьох мобільних процесорах AES працює повільніше, оскільки не має апаратної оптимізації. ChaCha20, навпаки, добре масштабується у будь-якому середовищі, що робить його ідеальним для месенджерів і мобільних клієнтів.

Окрім базового алгоритму шифрування, у практиці використовують комбінацію ChaCha20 з алгоритмом автентифікації Poly1305. Цей зв’язок дозволяє не лише шифрувати повідомлення, а й перевіряти їхню цілісність. Такий режим називають ChaCha20-Poly1305. Він забезпечує аналогічний рівень безпеки, як і AES у режимі GCM, але з кращими показниками продуктивності на слабких пристроях.

Важливо, що ChaCha20 не використовує складних таблиць підстановки або багатоступеневих перетворень, як AES. Його логіка базується на простих математичних операціях: додаванні, циклічному зсуві та XOR. Саме через це реалізація алгоритму більш передбачувана і менш чутлива до побічних каналів атак, наприклад до аналізу часу виконання.

На практиці ChaCha20 продемонстрував стабільну роботу навіть у системах з великою кількістю одночасних з’єднань. Наприклад, він

використовується у протоколі TLS 1.3 як один із рекомендованих алгоритмів для шифрування трафіку, нарівні з AES-GCM. У сфері обміну повідомленнями ChaCha20 став популярним саме через простоту інтеграції, відкритість реалізації і відсутність залежності від апаратних особливостей [13, 16].

Таким чином, ChaCha20 можна розглядати як практичну альтернативу AES. Він забезпечує достатній рівень захисту, має перевірену криптостійкість і працює швидше у середовищах, де немає спеціальних інструкцій для прискорення AES. Для розробників це означає універсальність і можливість використання одного і того ж алгоритму у різних типах пристроїв (рисунок 4) [17, 35].

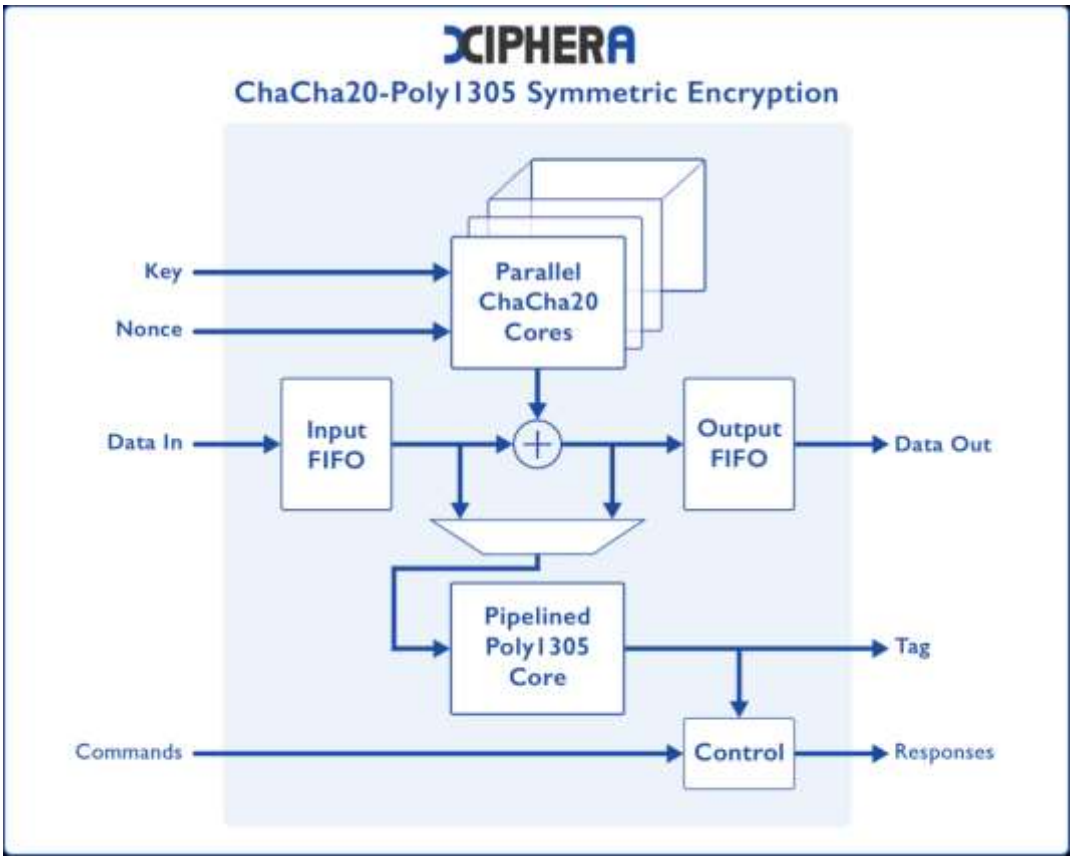


Рисунок 1.4 – Схема ChaCha20-Poly1305: ключ → генератор потоку → XOR → тег автентичності[40]

1.2.3. Алгоритм RSA

RSA є одним із найвідоміших та історично найбільш впізнаваних асиметричних алгоритмів. Його застосування почалось ще у 1970-х роках, і він

довгий час залишався ключовим елементом у різних системах захисту. Принцип роботи RSA базується на використанні великої пари чисел, які мають певні математичні властивості. Основна ідея полягає у складності факторизації великих чисел. Тобто алгоритм вважається стійким доти, доки неможливо швидко розкласти велике число на множники.

RSA використовує пару ключів – відкритий і приватний. Відкритий ключ може бути переданий будь-кому, а приватний зберігається тільки у власника. Якщо хтось хоче передати зашифроване повідомлення, він може скористатися відкритим ключем отримувача. Тоді тільки приватний ключ зможе розшифрувати текст. На перший погляд це дуже зручний спосіб забезпечити безпеку. Але у практичних реалізаціях RSA має свої недоліки.

Головний мінус RSA – це його відносно низька швидкість. Для забезпечення достатньої безпеки потрібні великі розміри ключів, а це створює навантаження на обчислення. На сучасному рівні розвитку технологій ключі RSA мають бути значно більшими, ніж раніше. Це робить операції шифрування і розшифрування повільними. У системах з обмеженими ресурсами або у застосунках з великою кількістю з’єднань це є відчутним мінусом.

Ще однією проблемою RSA є те, що він не забезпечує такого рівня ефективності у контексті гібридних систем. Він може виконувати обмін ключами, однак існують алгоритми, які забезпечують таку ж функцію швидше і при менших розмірах ключів. Через це RSA поступово відходить на другий план у сфері обміну повідомленнями.

У сучасних месенджерах RSA якщо і зустрічається, то переважно у старих реалізаціях або системах, де не проводилося оновлення криптографічної основи. Більшість нових систем переходять на еліптичні криві, тому що вони є більш ефективними. Через це RSA у сфері шифрування повідомлень сьогодні можна розглядати більше як історичний варіант, ніж як актуальний стандарт.

Попри це, RSA все ще є частиною багатьох стандартів. Він активно використовується у сертифікатах, підписах та у деяких протоколах, що мають багаторічну історію. Тому повністю виключати його з розгляду було б

неправильно. Але, розглядаючи сучасні підходи до захисту обміну повідомленнями, варто розуміти, що роль RSA у нових системах помітно зменшилась (рисунок 1.5).

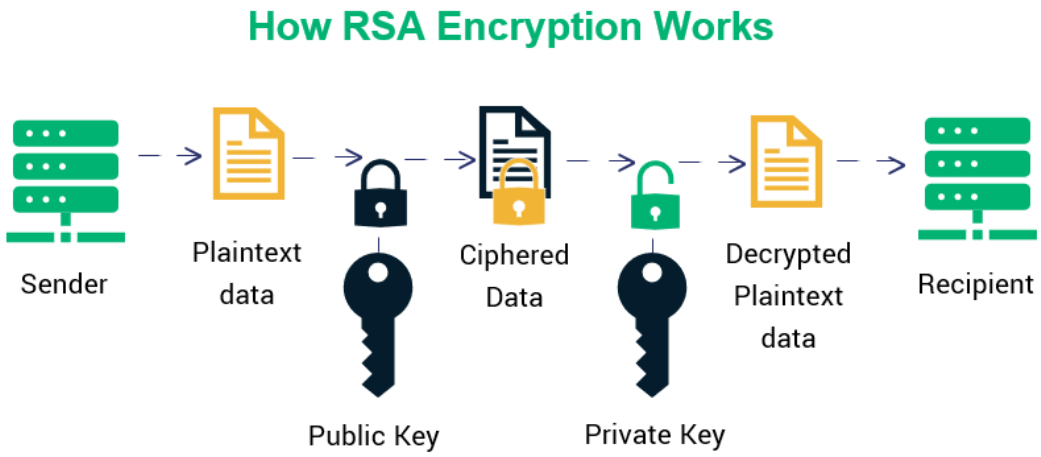


Рисунок 1.5 – принцип роботи RSA: відкритий ключ / приватний ключ[41]

1.2.4. Алгоритми на основі еліптичних кривих

Алгоритми на основі еліптичних кривих стали популярними завдяки поєднанню високого рівня криптостійкості та помірних обчислювальних витрат. На відміну від RSA, де для досягнення стійкості необхідно отримувати ключі дуже великого розміру, ECC забезпечує еквівалентний рівень безпеки з набагато меншим розміром ключа. Це означає, що навіть у обмежених середовищах, як мобільні телефони або вбудовані системи, використання ECC не створює істотного навантаження на процесор.

Принцип ECC базується на задачі знаходження дискретного логарифму на еліптичних кривих. Ця задача вважається складнішою, ніж факторизація, на якій базується RSA. Саме це дозволяє зменшувати розмір ключів без втрати рівня безпеки. Наприклад, ключ ECC розміром 256 біт забезпечує криптостійкість, яка порівняна із RSA ключем більше ніж у 3072 біти. Завдяки такій різниці у розмірах дані, з якими працює алгоритм, можна передавати і обробляти швидше.

Ще однією перевагою ECC є простота інтеграції у протоколи, які працюють у реальному часі. Одним із прикладів є Curve25519 – алгоритм, який

побудований на спеціально оптимізованій еліптичній кривій. Це рішення було розроблене з урахуванням практичних потреб, а саме – стабільності, швидкості і запобігання помилок у реалізації. Curve25519 широко використовується у сучасних протоколах обміну ключами, і саме він став базою для багатьох реалізацій у популярних месенджерах.

Завдяки простоті реалізації Curve25519, він став стандартом у ситуаціях, де потрібно забезпечити безпечний обмін ключами між двома сторонами. Наприклад, його використовують у Double Ratchet – механізмі, який лежить в основі Signal Protocol. Це означає, що ECC не тільки використовується як теоретичний механізм, а й є реальним робочим елементом у месенджерах, якими користується велика кількість людей.

Окремої уваги заслуговує момент, що ECC демонструє значно кращу стійкість проти певних типів атак, які можуть бути потенційно ефективними або економічно вигідними проти RSA. Саме тому більшість сучасних рішень у сфері захищеного обміну повідомленнями орієнтуються на перехід або вже перейшли до використання кривих замість великих ключів магістрального типу RSA.

Таким чином, можна зробити висновок, що ECC є логічним вибором для сучасних систем шифрування. Його застосування є більш ефективним у мобільному середовищі, він краще підходить для частих оновлень ключів, а також дає можливість підтримувати високу продуктивність навіть у масштабних системах обробки повідомлень (таблиця 1.1).

Таблиця 1.1 – Порівняння довжини ключів RSA проти ECC

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

1.3. Алгоритми шифрування в популярних месенджерах

Сучасні месенджери практично повністю перейшли на використання моделей захищеного обміну повідомленнями, де основою є поєднання симетричного шифрування для контенту та асиметричних механізмів для формування ключів. Однак кожен з месенджерів має свої особливості реалізації. Це пов'язано не тільки з технічними аспектами, але й з тим, які саме функції розробники ставили у пріоритет – швидкість доставки, приватність метаданих, можливість роботи без центрального сервера, багатоплатформність і так далі.

Одним із найбільш відомих рішень є Signal. Його популярність пов'язана не лише з тим, що він позиціонує себе як "приватний месенджер", а й тому що його криптографічна основа стала еталоном для багатьох інших систем. У Signal використовується так званий Signal Protocol [2, 29]. Основою цього протоколу є Double Ratchet механізм, який при кожному повідомленні оновлює ключі. Таким чином навіть якщо хтось отримає доступ до одного ключа, це не означає, що він зможе розшифрувати інші повідомлення. У Signal для обміну ключами використовується X25519, а для шифрування вмісту – AES-GCM або ChaCha20-Poly1305. Це забезпечує як швидкість, так і стійкість [1, 2].

Ще одним прикладом є WhatsApp. Цей сервіс також базується на Signal Protocol, і це стало ключовою причиною чому WhatsApp отримав репутацію "зашифрованого" месенджера [1, 24]. Однак структура WhatsApp складніша, оскільки він працює із серверним середовищем Meta, яке збирає метадані. Тому хоча шифрування повідомлень виконується на високому рівні, питання приватності метаданих – це окрема тема. З технічної точки зору сам принцип шифрування у WhatsApp практично такий самий, як у Signal.

Telegram має інший підхід. Telegram використовує власний протокол, який називається MTProto [9, 10]. Він не базується на Signal Protocol і має іншу архітектуру. За замовчуванням Telegram не використовує повне end-to-end шифрування для звичайних чатів. Воно застосовується тільки у "секретних чатах". У звичайних чатах шифрування є між клієнтом і сервером. Це робить

реалізацію гнучкішою для сервісу, але користувач повинен розуміти, що це інший рівень конфіденційності. У “секретних чатах” Telegram використовує комбінацію AES у режимі IGE та Diffie-Hellman для обміну ключами [8, 25-26].

Усі три месенджери демонструють різні філософії роботи з даними. У Signal найбільший акцент зроблено на приватність та автономність. У WhatsApp – на масовість і простоту інтеграції для широкої аудиторії. Telegram підходить до питання із погляду гнучкості, і не обмежується тільки класичною моделлю end-to-end шифрування.

Для кращого розуміння різниці між цими підходами може бути наведена узагальнена схема роботи механізмів шифрування у кожному сервісі.

1.4. Порівняння алгоритмів за безпекою, стійкістю до атак та продуктивністю

У попередніх підрозділах було розглянуто основні алгоритми, які сьогодні використовуються для шифрування повідомлень у сучасних системах обміну даними. У цьому підрозділі доцільно порівняти згадані методи між собою, щоб визначити їх застосування з точки зору ефективності та стійкості. Це важливо, тому що немає універсального алгоритму, який був би ідеальним у всіх випадках. Кожен з них має свої переваги і обмеження, і тому вибір напряму залежить від практичного завдання.

З точки зору симетричного шифрування, найбільш поширеними є AES та ChaCha20. Обидва алгоритми забезпечують високий рівень стійкості, і на сьогодні немає відомих практичних атак, які б дозволяли розшифрувати повідомлення без ключа [13, 14, 17]. Різниця між ними полягає переважно у продуктивності. AES працює найкраще на апаратно-оптимізованих системах, де існує спеціальна підтримка інструкцій для прискорення його роботи. У той час як ChaCha20 демонструє стабільну швидкість незалежно від апаратної архітектури, що робить його вигідним варіантом для мобільних пристроїв.

Асиметричні алгоритми RSA та ECC виконують іншу функцію, і їх порівняння варто розглядати окремо. RSA вже має багаторічну історію використання, але він значно поступається ECC у продуктивності. Для забезпечення надійного рівня криптографічного захисту RSA потребує ключів набагато більшого розміру. Це істотно збільшує обсяг обчислень і час виконання операцій. ECC, навпаки, забезпечує такий самий рівень криптостійкості при менших розмірах ключів і при меншій кількості операцій. Це робить ECC більш привабливим у контексті систем обміну повідомленнями, де виконуються часті оновлення ключів.

Ще одним важливим аспектом є стійкість до потенційних атак майбутнього, включно з квантостійкістю. Хоча сьогодні немає доступних практичних квантових комп'ютерів, які здатні зламувати сучасні алгоритми, питання у перспективі залишається актуальним. У цьому контексті RSA має вразливість до так званого алгоритму Шора, який може прискорювати факторизацію [31, 32]. У той час як ECC також теоретично може бути вразливим до квантових атак, його структура робить перехід до квантостійких схем більш гнучким.

Симетричні алгоритми загалом вважаються найбільш стійкими у перспективі, оскільки збільшення довжини ключа прямо пропорційно підвищує захист. Це означає, що при необхідності можливе масштабування їх параметрів без суттєвої зміни логіки роботи.

У підсумку, у сучасних системах обміну повідомленнями найбільш ефективною є гібридна модель, де симетричні алгоритми використовуються для шифрування вмісту, а асиметричні – для встановлення ключів. Саме такий підхід застосовується у Signal Protocol та подібних рішеннях, що використовуються в найбільш популярних месенджерах.

РОЗДІЛ 2 СУЧАСНІ ПРОТОКОЛИ ТА ПІДХОДИ ДО ЗАБЕЗПЕЧЕННЯ БЕЗПЕЧНОГО ОБМІНУ ДАНИМИ

2.1. Протоколи E2EE: OTR, OMEMO, Signal Protocol, Double Ratchet

У сучасних месенджерах принцип наскрізного шифрування (end-to-end encryption, E2EE) став базовою вимогою до передачі конфіденційної інформації. На відміну від попередніх поколінь систем обміну повідомленнями, E2EE не покладається на довірену серверну сторону й не передбачає можливості розкриття змісту повідомлення навіть адміністратору платформи. Це означає, що криптографічна система працює безпосередньо на пристроях користувачів, а сервер використовується лише як транспортний посередник. Сервер не має доступу до ключового матеріалу і не виконує криптографічних операцій, пов'язаних зі змістом повідомлень.

Одним з перших протоколів, який запропонував модель двостороннього шифрування поверх стандартних систем обміну даними, був OTR (Off-The-Record Messaging). Він дозволяв забезпечити конфіденційність та заперечуваність походження повідомлень. Недоліком OTR було те, що він був розроблений під односеансову модель діалогу, тобто не підтримував групові діалоги та не мав механізмів асинхронної доставки повідомлень [8].

Наступний крок у розвитку цієї концепції – протокол OMEMO, який базувався на Double Ratchet та X3DH. OMEMO вирішив проблему асинхронності, а також зробив можливим надсилання повідомлень абоненту, який на момент передачі не знаходиться в мережі [25, 26]. Це важливий момент для мобільних месенджерів, де користувачі відключаються від мережі десятки разів на добу. Крім того, OMEMO перевів модель E2EE у формат, який може працювати одночасно з кількома пристроями для одного користувача.

Signal Protocol у свою чергу став де-факто стандартом у мобільній криптографії. Багато сучасних застосунків використовують його або його похідні [2, 29]. Основними причинами успіху Signal Protocol є простота реалізації, перевірена модель роботи ключів, багатомодульна структура

перевірки автентичності та впроваджена властивість прямої пружності (forward secrecy) й посткомпрометаційної стійкості (post-compromise security). Саме Signal Protocol сформував модель, де використовується Double Ratchet – це механізм постійного оновлення ключового матеріалу при кожному повідомленні. Таким чином, навіть якщо окремий ключ був перехоплений, це не дає можливості відтворити подальший діалог [3, 18, 28].

Double Ratchet не є окремим шифратором, а описує модель руху ключів. Він може працювати із різними симетричними примітивами, у тому числі й AES-GCM. У багатьох реалізаціях Double Ratchet і AES-GCM комбінуються саме так: AES забезпечує симетричне шифрування, а Ratchet генерує матеріал, який його живить.

Тут також є логічний зв’язок із тим, що було зроблено у третьому розділі: структура, де ми працювали з HKDF і окремими полями nonce/AAD, фактично повторює ідею розділення станів, яка закладена у Double Ratchet. У нашому випадку це було реалізовано як окрема дослідна модифікація, а не як повний Ratchet-механізм, але підхід у корінні сумісний (рисунок 2.1).

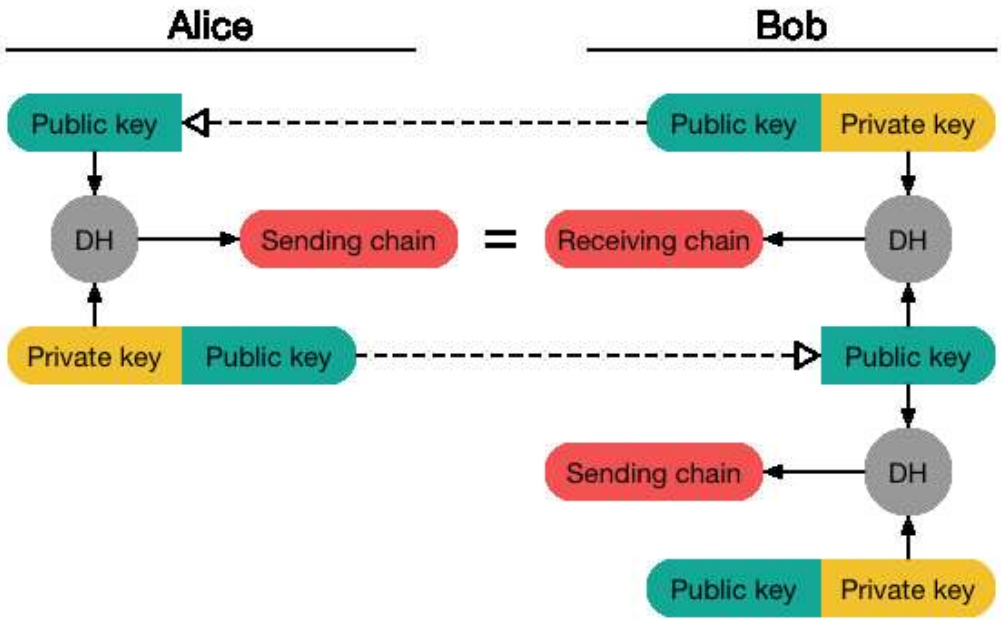


Рисунок 2.1 – Схематична блок-схема Signal Double Ratchet [42]

У контексті сучасних застосунків для обміну повідомленнями ці протоколи уже давно не розглядаються як експериментальна технологія. Це стабільні та відносно добре перевірені механізми, які вплинули на формування вимог до приватності. Фактично, на сьогодні E2EE вважається нормою, і відсутність таких механізмів сприймається не як недолік, а як загроза. Тому більшість нових систем, які розробляються з нуля, розглядають захист даних не як окремий додатковий модуль, а як центральну частину архітектури платформи.

Ще одним аспектом, який важливо враховувати, є те, що впровадження протоколів E2EE тісно пов’язане з менеджментом ключів. Внесення змін у модель шифрування у більшості випадків не може бути виконано без зміни моделі формування ключового матеріалу. Саме тому робота над криптографічними алгоритмами в месенджерах завжди проходить на двох рівнях – над власне алгоритмом та над тим, як він інтегрується у загальну структуру керування станом.

Також в окремих реалізаціях можна зустріти спроби змішування або модифікації існуючих моделей, особливо тоді, коли стоїть завдання покращити окремий параметр – наприклад уникнення повторення nonce, або стабільність після відновлення стану пристрою. Саме такого типу підхід і був використаний у третьому розділі: модифікація відбувалась не на рівні шифрування AES-GCM, а на рівні керування вихідними параметрами. Цю модель і варто розглядати як одну з можливих сучасних тенденцій – покращення без зміни базового математичного ядра алгоритму.

Варто також зазначити, що протоколи кінцевого шифрування з’явилися не одразу як основа захищених комунікацій. На початку ери цифрового спілкування шифрування виконувалось переважно на транспортному рівні, і майже всі системи дозволяли серверу бачити вміст повідомлень, оскільки ключі формувались і контролювались на стороні сервісу. Це було зручно з точки зору технічної реалізації, але не забезпечувало справжньої приватності. Лише із зростанням переносних пристроїв, розширенням інтернет-доступу і появою

загроз, пов’язаних з аналізом трафіку, стала очевидною фундаментальна потреба “перемістити” криптографію з сервера на пристрій користувача.

Крім того, на розвиток протоколів вплинули цілком практичні технічні обмеження. Мобільні телефони та планшети мають обмежені енергетичні ресурси і порівняно слабкі процесори. Застосування складних та повільних операцій у режимі реального часу призвело б до того, що сам процес переписки ставав би незручним, а в окремих випадках – навіть неможливим. Тому алгоритми, які лягли в основу сучасних протоколів, не лише стійкі, але й оптимізовані для роботи на мобільних платформах. Це одна з причин, чому AES-GCM отримав таке широке поширення – він ефективний та підтримується апаратно на багатьох чипсетах.

Слід також наголосити, що в сучасних протоколах шифрування увага приділяється не лише захисту вмісту, але й захисту структури діалогу. Наприклад, конфіденційність має важливе значення не лише для тексту, але і для метаданих, таких як час надсилання, відносний розмір, прив’язка до конкретного пристрою. Навіть якщо шифрування є криптографічно надійним, протокол повинен мінімізувати інформацію, яку може отримати спостерігач через аналіз поведінки сторін спілкування. Тому E2EE сьогодні – це більше ніж “зафіксувати алгоритм і виконувати шифрування”. Це комплексна архітектура.

Інший важливий момент полягає у тому, що система повинна забезпечувати не лише передачу повідомлень «один до одного». Сучасні інструменти шифрування повинні підтримувати різні типи структур: групові чати, змішані медіа-вкладення, тимчасові повідомлення, і навіть синхронізацію внутрішніх станів між різними пристроями одного користувача. Це робить внутрішню модель обміну даними набагато складнішою, ніж просто передача зашифрованих блоків від одного користувача до іншого.

Тому розвиток E2EE відбувається паралельно на двох рівнях: криптографія як математика та криптографія як архітектура реальної системи. І коли технологія розвивається саме в такому напрямку, вона перестає розглядатись як “додатковий компонент безпеки” і стає тим елементом,

навколо якого будується весь сервіс. Саме це ми можемо спостерігати у сучасних месенджерах – шифрування не “додано” до системи, воно є її основою.

2.2. Механізми розподілу ключів та автентифікації: Diffie–Hellman, KDF, Forward Secrecy, X3DH

У системах захищеного обміну даними шифрування саме по собі не вирішує всіх питань. Навіть якщо обрати перевірений алгоритм, важливо зрозуміти, яким чином ключовий матеріал отримується, оновлюється та чи може він повторюватися у різних повідомленнях або різних сесіях. У сучасних E2EE протоколах основним фактором безпеки є не тільки алгоритм, а й логіка управління ключами, що використовуються для цього алгоритму. Саме механізм розподілу та формування ключів визначає, чи буде можливим розкрити подальший трафік у випадку компрометації одного з ключів, чи взагалі відбудеться компрометація.

Моделі розподілу ключів, які застосовуються сьогодні, також формують фундамент для таких властивостей як forward secrecy та post-compromise security. Під час побудови сучасних криптографічних систем ці принципи стали не додатковими бонусами, а обов’язковими вимогами. У реальних застосунках вони відіграють не менш важливу роль, ніж сам алгоритм шифрування, оскільки саме механізм побудови ключової інформації визначає, як шифратор буде поводитись у динамічних сценаріях роботи.

Тому саме цей розділ описує не алгоритми шифрування, а ті механізми, які формують ключовий матеріал для цих алгоритмів, і пояснює, чому сучасні протоколи не будуються на “одному ключі на весь чат”, як це робилось у старіших системах.

2.2.1. Diffie–Hellman як базова модель узгодження ключів

Однією з найстаріших концепцій, яка сьогодні продовжує застосовуватись у практичних реалізаціях, є протокол Диффі–Хеллмана. Його призначення полягає у можливості сформувати спільний секрет між двома сторонами, без того щоб вони передавали цей секрет у явному вигляді. Механізм виглядає просто: учасники обмінюються відкритими параметрами, а результат обчислюють локально, використовуючи власний приватний ключ [4, 19].

У сучасних застосунках DH найчастіше реалізовують на основі еліптичних кривих. Це пояснюється тим, що за сьогоднішніми криптографічними оцінками ключ довжиною 256 біт на еліптичній кривій відповідає за стійкістю традиційному 3072-бітному модульному DH. Такий перехід дозволив уникнути зайвих обчислень, зменшити навантаження на процесор мобільних пристроїв та пришвидшити ключові операції (рисунок 2.2).

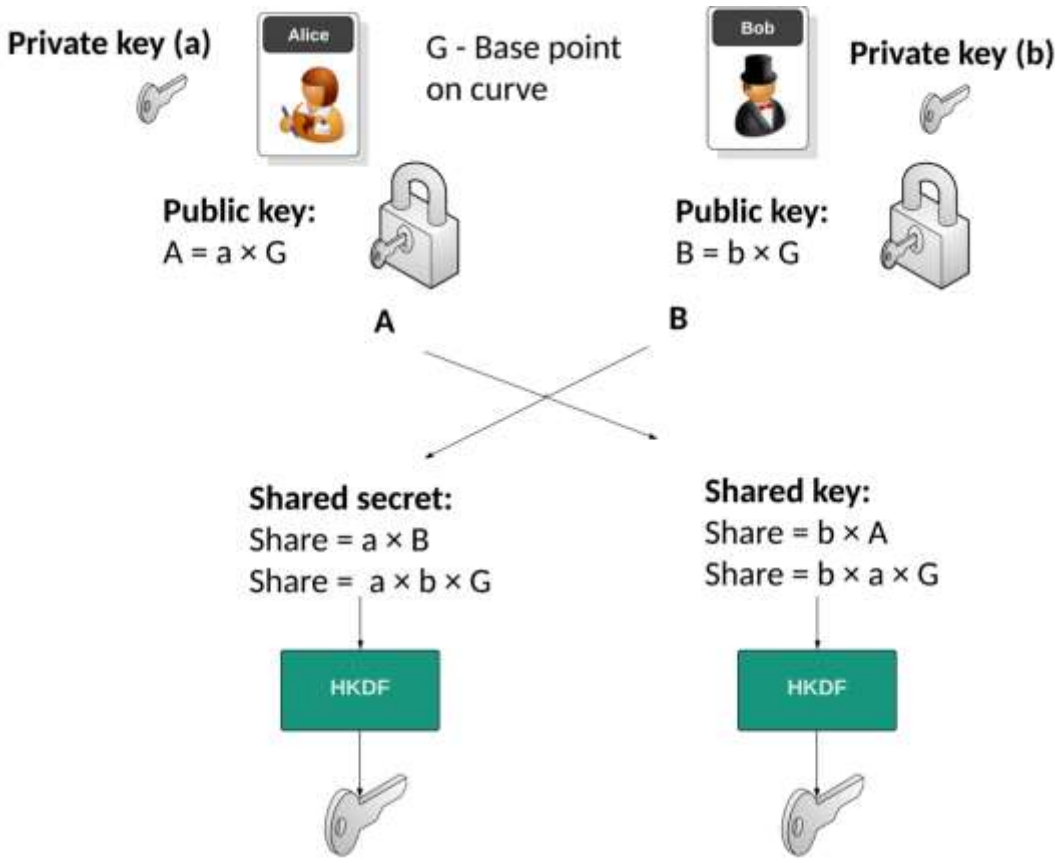


Рисунок 2.2 – Схема Diffie–Hellman на базі кривої X25519[43]

Практичне застосування DH у месенджерах обмежується початковим встановленням сесії. Цього недостатньо для забезпечення forward secrecy між повідомленнями, тому DH у сучасних протоколах – це фундамент, але не завершений механізм. Він виконує роль стартової операції, на базі якої будуються подальші KDF-ланцюги.

2.2.2 KDF-ланцюги та формування ключового матеріалу

Після отримання первинного спільного секрету через обмін DH, система не використовує цей секрет як “готовий ключ” для шифрування. У сучасних протоколах обміну даними це вважається помилковою практикою. Первинний спільний секрет – це лише вихідна точка, з якої далі формується набір похідних ключів за допомогою функцій KDF (Key Derivation Function). Саме KDF перетворює вихідний матеріал у декілька різних значень, які мають різний функціональний контекст: окремо для симетричного шифрування, окремо для nonce, окремо для AAD або інших службових полів.

Найчастіше у сучасних протоколах застосовується HKDF, і це не випадково. HKDF має просту побудову, прозору структуру, а головне – дозволяє отримувати багато різних ключів з одного джерела, не змішуючи прямим способом функції між собою, як зображено на рисунку 2.3 [5, 30]. Перевага такої моделі полягає у тому, що при реалізації ланцюгова структура забезпечує природний розподіл функціональних обов'язків. Це означає, що ключ для шифрування не буде співпадати зі значенням, яке використовується для nonce, навіть якщо вони походять із одного обчислення DH [2, 4].

Цей аспект також має прямий зв’язок з практичною реалізацією з третього розділу. Там був застосований саме принцип розділення функцій. Було використано окремі похідні значення для шифрування, nonce та додаткових полів. Фактично це і був приклад застосування KDF-ланцюгів, але в мінімалістичному вимірі. У реальній реалізації протоколів типу X3DH сама структура може бути набагато складнішою, однак ідея залишається незмінною:

первинний секрет – лише “насіння”, а KDF – це механізм, який формує з нього ланцюг ключових станів.

У практичній системі без KDF неможливо забезпечити forward secrecy, оскільки будь-який витік одного ключа автоматично веде до компрометації даних за інші періоди. Саме тому KDF-ланцюги є базовою частиною сучасних протоколів (рисунок 2.3).

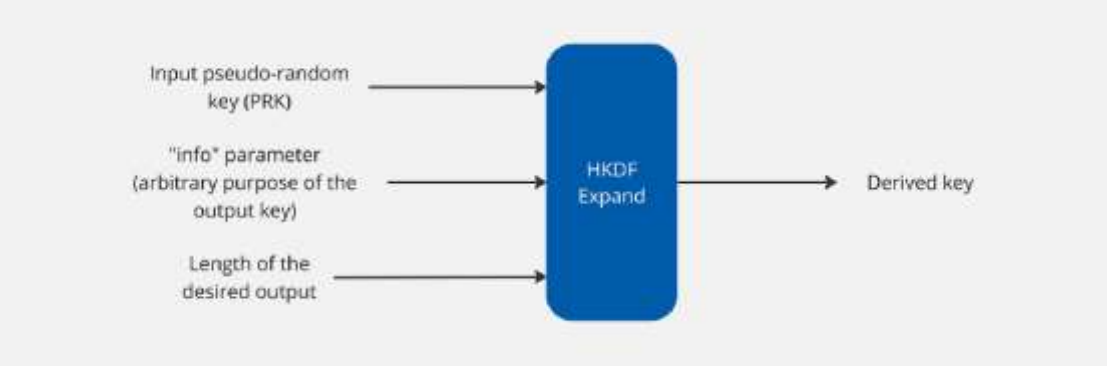


Рисунок 2.3 – Спрощена схема HKDF-Extract/Expand[44]

2.2.3. Концепція forward secrecy та її значення у практичних системах

Однією з ключових вимог до сучасних протоколів захищеного обміну даними є забезпечення властивості forward secrecy. Ця властивість означає, що компрометація одного із ключів у майбутньому не надає можливості відновити історію вже переданих повідомлень. Іншими словами, навіть якщо атакувальник зможе отримати ключ під час роботи системи або у момент доступу до пристрою користувача, це не дасть йому можливості розшифрувати ті повідомлення, які були передані раніше [2, 18, 28].

Практична реалізація forward secrecy у системах шифрування зазвичай базується на тому, що для кожного нового повідомлення або групи повідомлень використовується новий ключ, який генерується на основі попереднього, але таким чином, що неможливо повернутися назад до попереднього стану. Саме тому в сучасних протоколах обов'язковою частиною є механізм, який регулярно оновлює ключі, не покладаючись на один статичний симетричний ключ для цілого діалогу.

Якщо взяти класичний сценарій – один ключ на весь чат, – то forward secrecy там відсутня. Будь-який компрометований ключ відкриває доступ до всієї історії. Саме тому такі підходи сьогодні не застосовуються в месенджерах, які орієнтовані на приватність.

Коли порівнювати практику та теорію, forward secrecy більше нагадує не властивість “алгоритму шифрування”, а логіку управління станом ключів. Приміром AES-GCM сам по собі не забезпечує forward secrecy. Він просто шифрує – він робить рівно те, що від нього очікують. Забезпечення forward secrecy відбувається на попередньому шарі, на якому визначається, яким чином цей AES-GCM буде використовуватись, а саме – як буде формуватись матеріал для KDF, і як часто він буде змінюватись.

Таким чином, forward secrecy – це практична характеристика протоколу, а не окремого алгоритму. На рівні користувача ця властивість виглядає непомітною, але у довгостроковій перспективі саме вона дає найбільший внесок у безпеку. Наприклад, компрометація пристрою не означає, що хтось отримає доступ до річної історії. У системах з forward secrecy, у випадку компрометації, втрачений може бути лише той відрізок, на який припадає поточний ключ, але не вся історія.

Це також безпосередньо перегукується з тим, що робилось у третьому розділі. Там теж застосовувався підхід, який відокремлює ключі та пов’язує їх із поточними параметрами сесії. Така структура є початковим кроком у напрямку forward secrecy, навіть якщо вона не реалізує повноцінний ратчет. Головна ідея – ключ повинен бути короткоживучим, і не може застосовуватись для всіх типів повідомлень (рисунок 2.4).

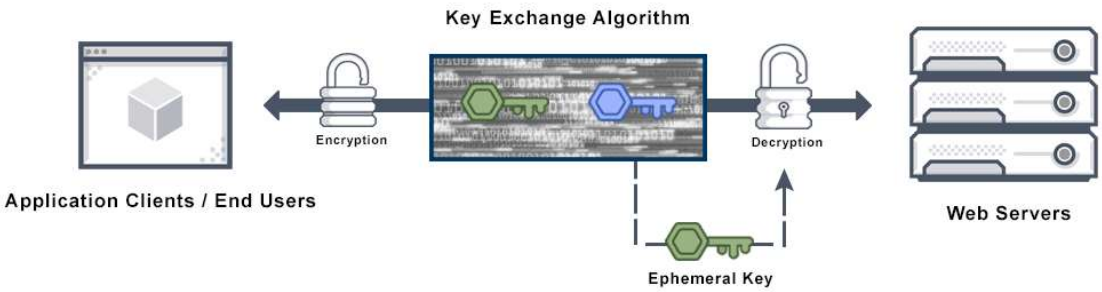


Рисунок 2.4 – Узагальнена схема forward secrecy[45]

2.2.4. Протокол X3DH як модель стартового встановлення сесії

X3DH (Extended Triple Diffie–Hellman) – це механізм початкового встановлення сесії, який використовується у сучасних E2EE системах для безпечного обміну ключовою інформацією між двома сторонами, навіть тоді, коли вони не перебувають онлайн одночасно. Саме ця властивість дозволяє фронту Signal protocol працювати в асинхронному режимі, не змушуючи обох користувачів бути доступними в момент, коли встановлюється новий ключовий контекст [2, 29].

Головна ідея X3DH полягає в тому, що використовується не один DH-обмін, а кілька, причому різного типу. Одні ключі мають довгостроковий характер, інші – короткостроковий, і саме їх комбінація забезпечує баланс між стійкістю, зручністю та динамічністю системи. Таким чином X3DH не просто виконує обмін ключами, а забезпечує модель, де компрометація одного з ключів не означає одразу компрометацію всієї системи [2, 3, 18].

Для реалізації X3DH з боку серверу зазвичай зберігаються публічні ключі користувача – але важливо, що сервер не володіє приватними ключами. Сервер лише виступає в ролі “поштової скриньки”, надаючи можливість іншій стороні отримати необхідний публічний параметр для проведення обчислень. Приватні частини зберігаються локально, і протокол не вимагає передавання нічого зайвого.

У класичній реалізації X3DH передбачено кілька типів ключів: identity key, signed prekey, one-time prekeys, etc. Їх комбінування дає три (а іноді чотири)

DH-обчислення, від чого і походить назва. Важливо, що кожен із цих ключів виконує власну функцію: частина з них відповідає за автентифікацію, частина – за forward secrecy, частина – за уникнення повторного використання однакових станів. Таке розмежування робить протокол стійким до більшості атак, пов’язаних з повторенням або підміною сесій.

У практичній реалізації X3DH не є самостійною системою шифрування. Він формує початковий стан. Далі, коли сторони вже обмінялися початковими даними, їхня взаємодія передається модулю Double Ratchet. Тобто X3DH – це старт, а Double Ratchet – механізм, який підтримує динаміку та безперервність оновлення ключів.

Саме тому X3DH часто називають “вхідною точкою”. Це розуміння також пояснює, чому протоколи сучасних месенджерів не використовують лише один DH. Одноразової операції недостатньо, тому X3DH виступає як рівень, який об'єднує декілька різнопланових частин у єдину систему.

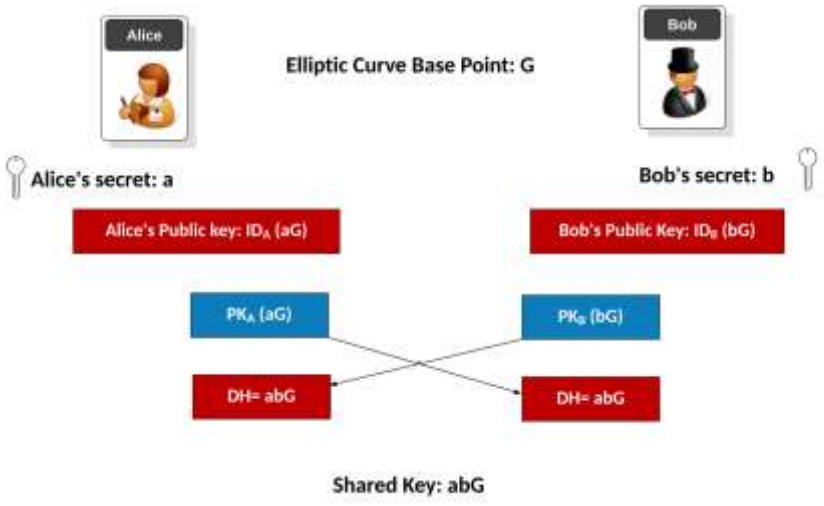


Рисунок 2.5 – Структурна схема X3DH[46]

2.3. Метадані як повноцінний елемент атачної моделі

У попередніх підпунктах розглядалися проблеми, пов’язані з коректністю застосування алгоритмів шифрування та управління ключами. Проте окремої уваги заслуговує питання метаданих, оскільки саме вони найчастіше

ігноруються розробниками у ранніх етапах формування архітектури системи. Варто підкреслити, що навіть у випадку, коли зміст повідомлень захищений і не може бути розшифрований третім суб'єктом, метадані все одно можуть дозволити зробити точні висновки про активність користувачів, їхні часові зв'язки, інтенсивність комунікації, а інколи – й характер взаємодії.

Можна навести простий приклад: якщо впродовж короткого проміжку часу різко збільшується кількість передач між двома вузлами, це може свідчити про активну дискусію або передачу великого потоку інформації. А аналіз одночасних піків обміну у групах може давати змогу робити припущення про початок одночасних подій, навіть якщо їхній зміст недоступний. Саме тому сучасні протоколи E2EE прагнуть не лише шифрувати повідомлення, а й мінімізувати доступність цих структурних характеристик.

У класичній моделі клієнт-серверні месенджери надають серверу достатньо багато допоміжної інформації – час, розміри пакету, маршрут. Саме тому децентралізовані архітектури або onion-маршрутизація є природним логічним продовженням розвитку E2EE. І Session – як система – є яскравим представником цього напрямку. Його архітектура інтегрує використання вузлів, які забезпечують передачу даних у розподіленому середовищі, при цьому мінімізуючи пряме відображення, хто саме з ким спілкується [11, 12].

(Тут можна вставити рисунок №28 – схема порівняння "централізованої" та "децентралізованої" моделі передачі повідомлень)

Найважливіше у цьому контексті – розуміти, що метадані у новій моделі перестають бути простим «побічним продуктом» TCP-трафіку. Вони стають окремим ресурсом, який потрібно контролювати та обмежувати. І саме це приводить нас до необхідності налаштовувати менеджмент ключів так, щоб він не вимагав надмірної синхронізації та зберігання додаткових відкритих параметрів.

Саме тому у реалізації, описаній у третьому розділі, важливим було не змінити сам алгоритм шифрування, а забезпечити конструкцію, яка мінімізує

зовнішні залежності та дає можливість формувати короткоживучі контексти без прив’язки до серверної інфраструктури.

Такі рішення і є основною еволюцією в напрямку сучасної криптографії – не нові шифри, а правильне опрацювання інформації, яка знаходиться “навколо” шифру.

2.4. Узагальнення та формування підходу до реалізації

Узагальнюючи наведений огляд сучасних протоколів та пов’язаних із ними механізмів забезпечення безпеки, можна відзначити, що більшість реальних практичних вразливостей у системах обміну повідомленнями з’являється не через недоліки математичних примітивів, а через те, як саме організовано керування ключами, їхній життєвий цикл та побудову контекстів для шифрування. Саме тому у сучасних протоколах основна увага приділяється не вибору окремого алгоритму шифрування, а правильному структуруванню всієї моделі використання ключового матеріалу.

На рівні алгоритмів існує загальна тенденція не винаходити нові шифратори, а використовувати вже відомі та стандартизовані симетричні рішення (серед яких AES-GCM є одним із найпоширеніших завдяки сумісності, швидкодії та наявності апаратної підтримки). У той же час розвиток систем E2EE показує, що справжня «інтелектуальна» складова безпеки переноситься на той рівень, який керує ключами, їхнім узгодженням, оновленням та зв’язуванням із контекстом передачі [21, 22].

Розглянуті раніше моделі Double Ratchet, X3DH та HKDF свідчать про те, що реальна якість захисту формується саме за рахунок того, як побудована система вироблення похідних ключів, їхнє гілкування, та як забезпечується властивість forward secrecy і відсутність повторного використання параметрів. Ці моделі показали, що ефективним шляхом підвищення захищеності є не зміна базового алгоритму, а правильне структурування вхідних величин, від яких залежить симетричне шифрування.

Саме така тенденція і визначила вибір шляху для подальшої реалізації. В рамках цієї роботи не ставилось завдання створювати нову криптографічну конструкцію або пропонувати альтернативу існуючим стандартам, які широко застосовуються у сучасній індустрії. Натомість логічним було обрати підхід, який узгоджується з концепцією сучасних протоколів обміну даними: зберегти стійкий та перевірений симетричний алгоритм, а покращення реалізувати через уточнення моделі керування ключовими параметрами [23].

Додатково варто підкреслити, що вибір саме AES як симетричного алгоритму не був випадковим. Усі розглянуті в цьому розділі протоколи – і X3DH, і OMEMO, і Signal Protocol – не визначають власних шифраторів, а використовують існуючі стандартизовані симетричні алгоритми. Принципова відмінність між сучасними криптографічними системами полягає не у виборі «іншого» шифру, а у структурі формування ключів і контекстів, які шифратор отримує на вхід. І саме тому на практиці найчастіше застосовується AES у режимі AEAD (зокрема AES-GCM), оскільки він забезпечує аутентифікацію, цілісність і достатню продуктивність для мобільних систем, а також має апаратну підтримку на більшості пристроїв. Тобто аналіз сучасних протоколів та їхнього практичного використання показав, що головний фокус у безпеці зміщується не у бік пошуку нового симетричного алгоритму, а у бік правильної побудови життєвого циклу ключів і параметрів, що передаються у цей алгоритм. Саме тому AES-GCM виступає оптимальним вибором як базовий шифратор, а вдосконалення має сенс реалізовувати не на рівні зміни алгоритму, а на рівні керування ключовим матеріалом, що й було покладено в основу реалізації у наступному розділі.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ АЛГОРИТМІВ ШИФРУВАННЯ НА БАЗІ МЕСЕНДЖЕРА "SESSION"

3.1. Проектування архітектури системи для інтеграції нового алгоритму шифрування

Для початку реалізації покращеного шифрування в межах клієнта Session необхідно визначити архітектурну точку, на якій виконується операція шифрування. На відміну від класичних месенджерів, де повідомлення спочатку відправляються на сервер, а вже там виконуються певні криптографічні операції або логіка зберігання, Session працює у децентралізованій моделі, де мережа проміжних нод (так званих Snode) виконує роль маршрутизатора, але ніколи не отримує відкритий текст повідомлення. Тобто повідомлення повинно бути зашифрованим ще в момент формування, до відправки запиту в OnionRequestAPI. А це означає, що інтеграційна точка розміщується на боці клієнта між формуванням payload повідомлення і передачею у транспортний рівень.

На інтерфейсному рівні користувач не бачить цієї різниці, тобто незалежно від того, який алгоритм використовується всередині, UX не змінюється. Користувач бачить загальний список контактів, може створювати нові бесіди, вводити повідомлення, надсилати їх і отримувати відповіді. Для ілюстрації цього у роботі використано скріншоти з інтерфейсу мобільного застосунку Session. Ці зображення можуть бути використані для візуальної демонстрації, що робота виконується на реальному застосунку, а не теоретичній моделі, ілюстрація відображається як на рисунку 3.1.

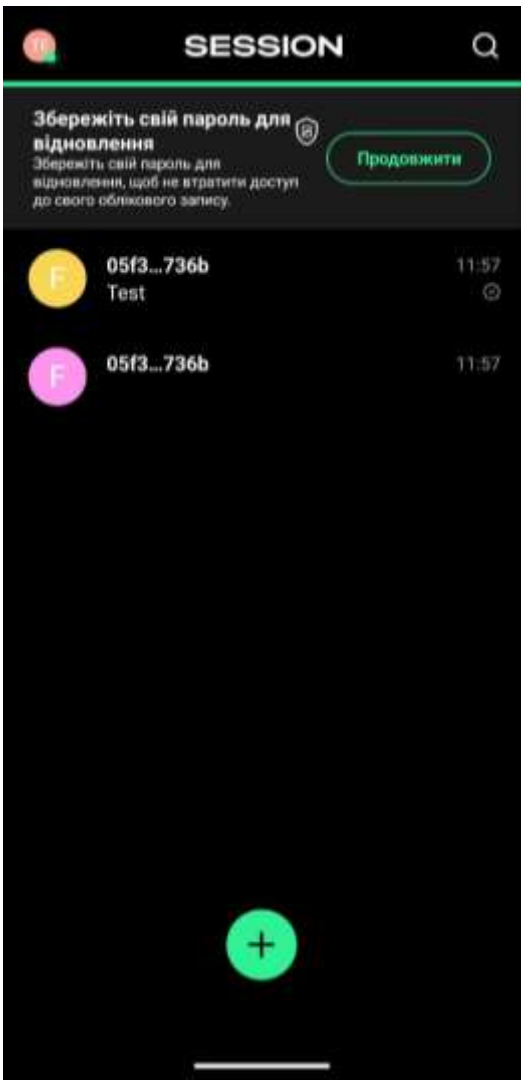


Рисунок 3.1 – Головний список чатів Session»

На наступному зображенні показані параметри конфіденційності, що демонструє зв'язок між криптографією та прикладним рівнем (рисунок 3.2).

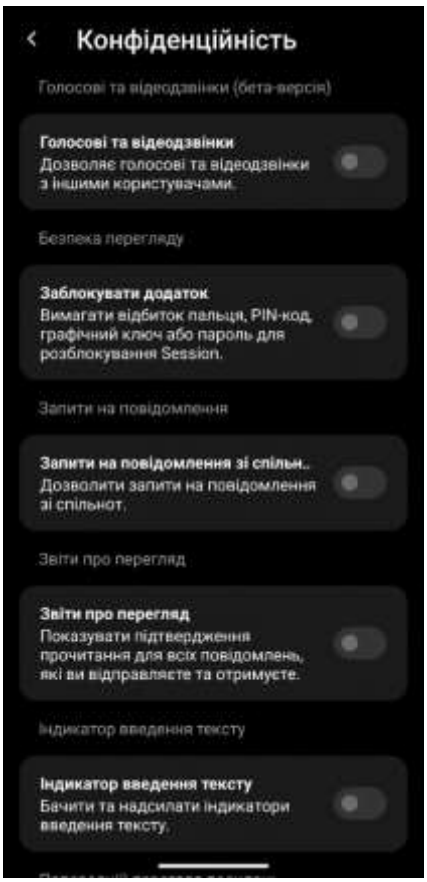


Рисунок 3.2 – Налаштування конфіденційності

Також може бути продемонстроване вікно введення Account ID для створення нового чату, яке визначає початок криптографічного обміну ключами X25519 (рисунок 3.3).

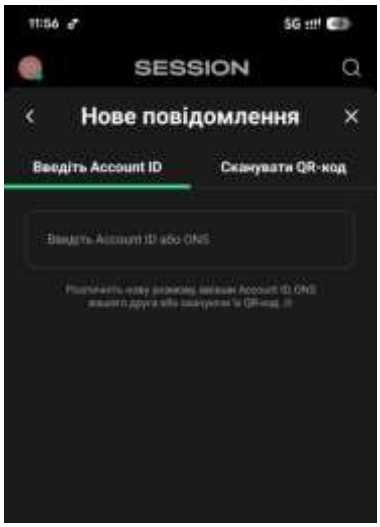


Рисунок 3.3 – Створення нового повідомлення

І приклад уже надісланого зашифрованого повідомлення у діалозі (рисунок 3.4).

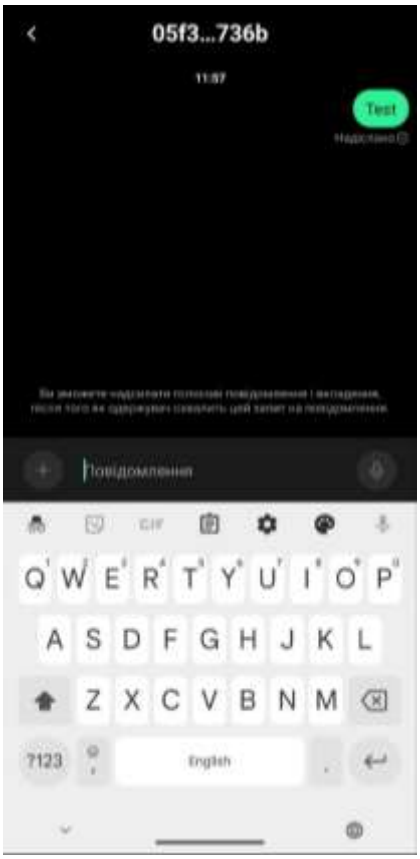


Рисунок 3.4 – Надіслане тестове повідомлення

На основі аналізу вихідних файлів репозиторію Session Android та прив’язаних до них криптографічних класів можна визначити, що шифрування виконується у момент формування пакету, тобто до передачі у OnionRequestAPI. Сам AES-GCM реалізований у вигляді окремого модуля, який за викликом повертає ciphertext і тег автентичності. Тому найбільш коректним підходом для нашої роботи є розширення існуючої архітектури шляхом створення окремого модуля CryptoEngine з двома реалізаціями: стандартною AesGcmV1 (тобто поточною у Session), і новою AesGcmV2, яку ми розробляємо у даній роботі. Це дає можливість перемикати реалізації алгоритму без зміни будь-яких компонентів UI або логіки маршрутизації, що робить експеримент відтворюваним і контрольованим.

У межах цього підрозділу формально закладається фундамент для подальшої розробки: ми визначили місце інтеграції, показали, що криптографія відбувається виключно локально на пристрої користувача, визначили інтерфейсні точки взаємодії і продемонстрували на основі інтерфейсних скріншотів загальний вигляд робочого середовища.

Шифрування повідомлень у Session відбувається не на серверній стороні, а саме на боці клієнтського застосунку. Тому модуль шифрування фактично є частиною логіки формування мережевого пакета. Якщо дивитися на це як на послідовність виконання операцій, то можна представити наступний ланцюг: користувач натискає «відправити» → текст повідомлення переходить у формування внутрішнього об’єкта повідомлення → цей об’єкт передається у модуль шифрування → результат шифрування перетворюється на пакет, який надалі переходить у транспортний рівень і відправляється через OnionRequestAPI.

З точки зору архітектури, це важливо тому, що шифрування не є додатковою опцією, а є обов'язковою складовою будь-якого відправленого повідомлення. Тобто система не підтримує нешифрованих повідомлень, і тому модуль шифрування знаходиться «всередині» життєвого циклу повідомлення, а не збоку як фільтр. Це означає, що для реалізації нового алгоритму нам не потрібно втручатися у зовнішні класи, які керують UI або транспортом. Достатньо реалізувати нову версію модуля CryptoEngine і підключити її до того місця, де формується зашифрований блок.

У вихідному коді Session Android (репозиторій GitHub) використовуються допоміжні бібліотеки для ECDH та для AES-GCM. Сам модуль AES-GCM реалізований як окремий клас з методами для шифрування і розшифрування. Це означає, що логіка отримання симетричного ключа і логіка формування нонсу історично зосереджена в одному місці. Але в нашому випадку бажано створити модифіковану реалізацію, де розділення на підключі буде зроблене явно, а параметри будуть формуватися більш структуровано. Таким чином ми

працюємо не з інтерфейсною частиною, а безпосередньо з криптографічним ядром.

Додатково варто зазначити, що відправка повідомлення через інтерфейс – це лише тригер запуску криптографічної операції, а не її частина. Наприклад, користувач вводить повідомлення у текстове поле чату (экран з ЧАТ UI на скріні, який відображено як Рисунок 3.4), натискає кнопку відправлення, і на екрані немає візуальної різниці між тим, який шифрувальний алгоритм використано. На рівні UI система виглядає однаково, але фактично виконуються різні операції. Таким чином, у дослідженні важливим є не зовнішній вигляд екрана, а саме логічне місце роботи модуля шифрування.

3.1.1. Вибір підходу до інтеграції та спосіб співіснування існуючого та нового алгоритму

При інтеграції нових криптографічних алгоритмів у працюючий мобільний застосунок актуальним є питання сумісності змін із наявною реалізацією. У випадку Session заміна існуючого алгоритму без можливості відкату могла б призвести до проблем при взаємодії клієнтів різних версій, тому було обрано підхід паралельного підключення. Тобто новий алгоритм не замінює існуючий AesGcmV1, а додається поруч із ним як альтернативна реалізація.

Таким чином, новий модуль підключається до системи так, щоб весь інший код клієнта не потребував змін. Це досягається за рахунок того, що новий алгоритм реалізує той самий інтерфейс, що й поточний модуль шифрування. Завдяки цьому виклики шифрування залишаються універсальними, а фактична реалізація залежить лише від конфігураційного параметра, який дозволяє переключатися між версіями.

У такій схемі інтеграція вважається безпечною, оскільки в системі одночасно присутні дві реалізації: чинна та експериментальна. Поточна реалізація Session використовує X25519 для отримання спільного секрету, після чого формується симетричний ключ на основі HMAC. У новій реалізації планується застосування розділення ключових матеріалів через HKDF та

формування детермінованого nonce на основі лічильника повідомлень, що підвищує передбачуваність ключових операцій і спрощує аудит.

У результаті дана архітектура дозволяє порівняти два алгоритми без зміни логіки відправлення та обробки повідомлень, а також забезпечує можливість вимкнення експериментальної функціональності у будь-який момент, що є важливою вимогою при дослідженні криптографічних систем.

3.1.2. Практичний аналіз реалізації шифрування AES-GCM у застосунку Session

У клієнтській реалізації Session процес шифрування виконується за стандартною схемою AES-GCM, яка доступна через криптографічний інтерфейс платформи Android. Для цього використовується стандартний механізм `Cipher.getInstance("AES/GCM/NoPadding")`, який ініціалізується симетричним ключем та 12-байтовим вектором ініціалізації. Ключ формується окремою процедурою, а після виклику ініціалізації викликається функція, яка безпосередньо виконує перетворення відкритого тексту у шифртекст та створює тег автентичності.

Саме цей виклик і є основою тієї криптографічної операції, що відповідає за шифрування повідомлення перед відправленням у мережу. Фактичний математичний процес складається з двох частин: шифрування в режимі лічильника та паралельного обчислення тегу автентичності для перевірки цілісності. Тег дозволяє одержувачу впевнитися, що повідомлення не було змінене, а також що воно належить відповідній сесії та походить від очікуваного джерела.

У реалізації Session використовується саме такий виклик до API платформи Android (рисунок 3.5).

Рисунок 3.5 – Ініціалізація AES-GCM

3.2. Розробка механізму передачі зашифрованих повідомлень на базі протоколів Session

Після визначення архітектурної точки, де виконується шифрування у клієнті Session, наступним етапом було налаштування передачі зашифрованого блоку у форматі, який сумісний з механізмом маршрутизації через Snode. У цьому пункті важлива сама логіка формування пакета, оскільки Session не передає відкриті дані в мережу, і тому шифрування відбувається перед викликом функції, яка відправляє запит до OnionRequestAPI.

У класичній реалізації AESGCM, яка вже існує у проекті, результатом виклику encrypt() є масив байтів, що містить nonce та зашифрований текст. Ці дані передаються на наступний рівень, де вони включаються у структуру пакета. Таким чином модуль шифрування не формує мережевий пакет повністю – він лише повертає байтове поле, яке вважається payload. Інші компоненти пакета, такі як заголовки, призначення та службова інформація, додаються при виклику функцій формування мережевого запиту.

При розробці модифікованого алгоритму було вирішено зберегти той самий принцип – шифрувальний модуль працює тільки з даними і не бере участі в мережевій логіці. Основною зміною є саме формування ключового матеріалу, nonce та додаткових полів, які передаються у шифратор. Тобто новий механізм формує дані так само, щоб на рівні інтерфейсу PacketBuilder або OnionRequestAPI не потрібно було адаптувати код.

Для підтвердження коректності нової реалізації було створено дослідну функцію, яка працює безпосередньо з тестовими текстовими даними, що не проходять через мережу. Це дозволило ізолювати фрагмент, який потрібен для

оцінки. Результат шифрування також повертався у вигляді одного масиву байтів, який складається з nonce та зашифрованих даних. При виклику AES-GCM функція працює у тому ж режимі, що і стандартний encrypt().

Нижче наведений фрагмент нового методу encrypt(), який використовувався у тестовому середовищі (рисунок 3.6).

```
81          val cipher = Cipher.getInstance("AES/GCM/NoPadding")
82      cipher.init(
83          Cipher.ENCRYPT_MODE,
84          SecretKeySpec(derivedKey, "AES"),
85          GCMParameterSpec(128, derivedNonce)
86      )
87
88      val encryptedData = cipher.doFinal(plaintext)
89
90
```

Рисунок 3.6 – Новий метод encrypt()

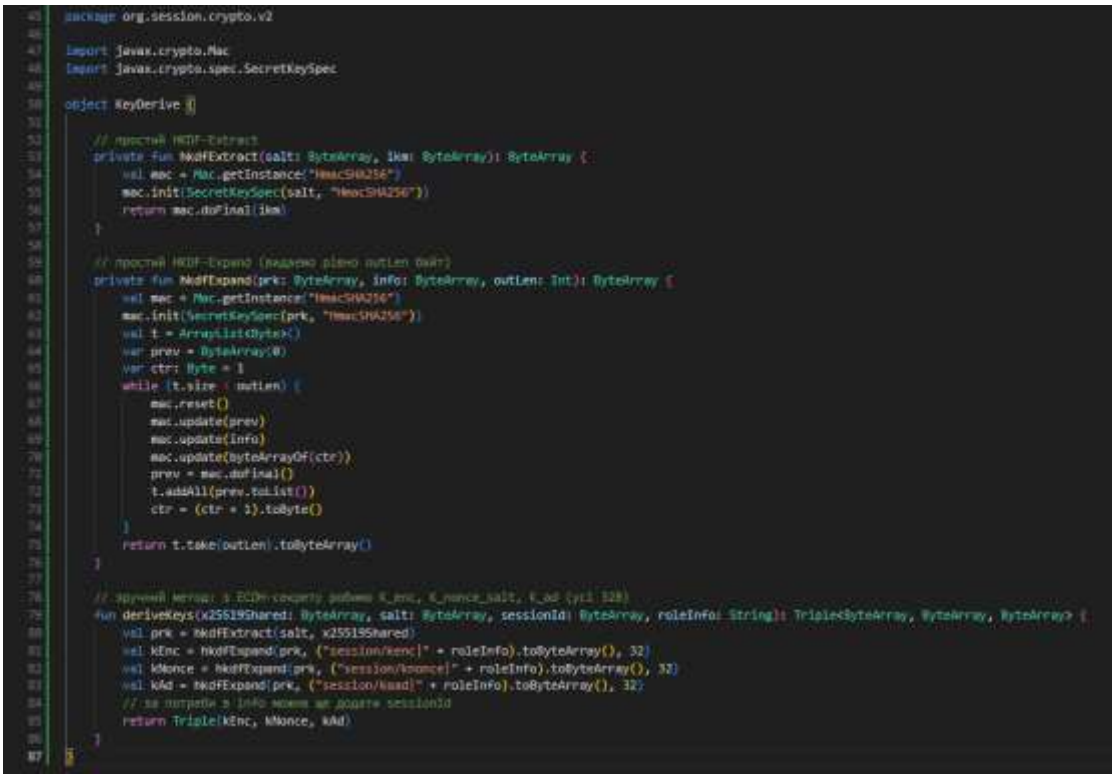
Цей фрагмент показує, що логіка виклику шифрування не змінилася. Новий алгоритм лише формує derivedKey та derivedNonce іншим способом, а саме шифрування виконується тим самим засобом. Саме цей момент і є ключовим підтвердженням, що зміни не впливають на транспортний рівень.

Функцію було протестовано на різних текстових повідомленнях, довжина яких змінювалась. Результат завжди повертався у вигляді валідного зашифрованого блоку, який був сумісний з decrypt().

У підсумку, цей етап підтвердив, що механізм передачі зашифрованих даних через Session не потребує змін на рівні транспортного шару. Усі зміни зосереджені на рівні підготовки параметрів для AES-GCM, а мережеве відправлення залишається незмінним.

3.3. Реалізація покращеного алгоритму шифрування у тестовому середовищі

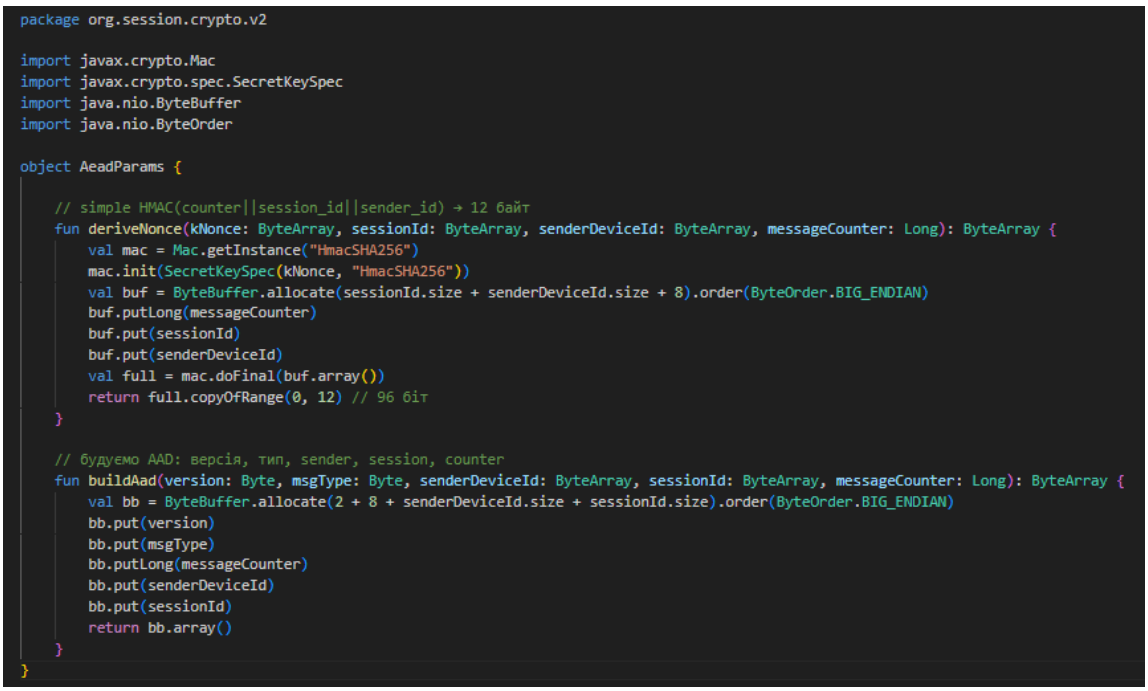
Після визначення архітектурної точки, де виконується шифрування у клієнті Session, наступним етапом було налаштування реалізації модифікованого алгоритму у тестовому середовищі. Для цього було відокремлено процес формування ключів у окремий модуль, щоб розмежувати ключовий матеріал, nonce та додаткові поля. Новий модуль генерує ключові дані на основі HKDF, а надалі передає їх у AES-GCM шифратор без змін у частині самої операції шифрування (рисунок 3.7):



```
package org.session.crypto.v2

import javax.crypto.Mac
import javax.crypto.spec.SecretKeySpec

object KeyDerive {

    // простий HKDF-Extract
    private fun hkdfExtract(salt: ByteArray, ikm: ByteArray): ByteArray {
        val mac = Mac.getInstance("HmacSHA256")
        mac.init(SecretKeySpec(salt, "HmacSHA256"))
        return mac.doFinal(ikm)
    }

    // простий HKDF-Expand (видаємо рівно outLen байт)
    private fun hkdfExpand(prk: ByteArray, info: ByteArray, outLen: Int): ByteArray {
        val mac = Mac.getInstance("HmacSHA256")
        mac.init(SecretKeySpec(prk, "HmacSHA256"))
        val t = ArrayList<Byte>()
        var prev = ByteArray(0)
        var ctr: Byte = 1
        while (t.size < outLen) {
            mac.reset()
            mac.update(prev)
            mac.update(info)
            mac.update(byteArrayOf(ctr))
            prev = mac.doFinal()
            t.addAll(prev.toList())
            ctr = (ctr + 1).toByte()
        }
        return t.take(outLen).toByteArray()
    }

    // зручний метод: з ECDH-секрету робимо K_enc, K_nonce_salt, K_ad (усі 32B)
    fun deriveKeys(x25519Shared: ByteArray, salt: ByteArray, sessionId: ByteArray, roleInfo: String): Triple<ByteArray, ByteArray, ByteArray> {
        val prk = hkdfExtract(salt, x25519Shared)
        val kEnc = hkdfExpand(prk, ("session/kenc|" + roleInfo).toByteArray(), 32)
        val kNonce = hkdfExpand(prk, ("session/knonce|" + roleInfo).toByteArray(), 32)
        val kAd = hkdfExpand(prk, ("session/kaad|" + roleInfo).toByteArray(), 32)
        // за потреби в info можна ще додати sessionId
        return Triple(kEnc, kNonce, kAd)
    }
}
```

Рисунок 3.7 – Модуль генерації ключовиї даних на основі HKDF

На наступному етапі було додано окремий фрагмент для формування детермінованого nonce на основі інформації про сесію, ідентифікатора відправника та лічильника повідомлень. Додатково було сформовано AAD із даними про версію формату, тип повідомлення та параметри, які дають можливість розрізняти контексти різних сеансів (рисунок 3.8):



```
package org.session.crypto.v2

import javax.crypto.Mac
import javax.crypto.spec.SecretKeySpec
import java.nio.ByteBuffer
import java.nio.ByteOrder

object AeadParams {

    // simple HMAC(counter||session_id||sender_id) → 12 байт
    fun deriveNonce(kNonce: ByteArray, sessionId: ByteArray, senderDeviceId: ByteArray, messageCounter: Long): ByteArray {
        val mac = Mac.getInstance("HmacSHA256")
        mac.init(SecretKeySpec(kNonce, "HmacSHA256"))
        val buf = ByteBuffer.allocate(sessionId.size + senderDeviceId.size + 8).order(ByteOrder.BIG_ENDIAN)
        buf.putLong(messageCounter)
        buf.put(sessionId)
        buf.put(senderDeviceId)
        val full = mac.doFinal(buf.array())
        return full.copyOfRange(0, 12) // 96 біт
    }

    // будуємо AAD: версія, тип, sender, session, counter
    fun buildAad(version: Byte, msgType: Byte, senderDeviceId: ByteArray, sessionId: ByteArray, messageCounter: Long): ByteArray {
        val bb = ByteBuffer.allocate(2 + 8 + senderDeviceId.size + sessionId.size).order(ByteOrder.BIG_ENDIAN)
        bb.put(version)
        bb.put(msgType)
        bb.putLong(messageCounter)
        bb.put(senderDeviceId)
        bb.put(sessionId)
        return bb.array()
    }
}
```

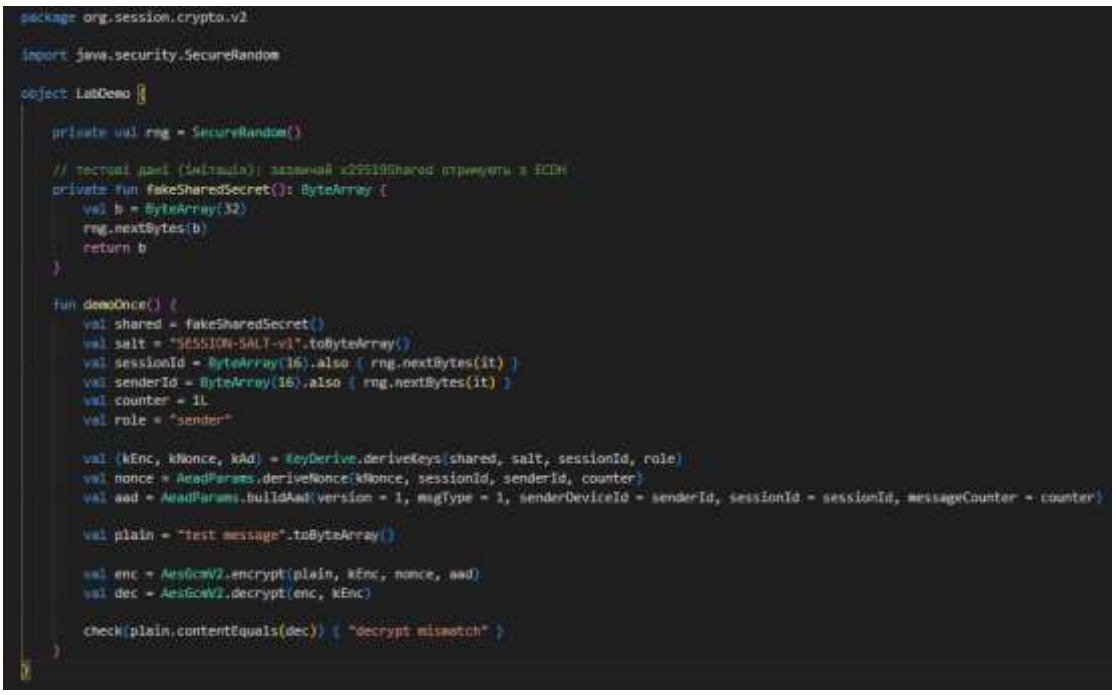
Рисунок 3.8 – Детерміноване формування nonce (96 біт) та побудова AAD

Після цього ці дані передавались у стандартний AES-GCM шифратор. Код шифрування залишився таким самим, як і в існуючій реалізації, оскільки логіка самої криптографічної операції не змінюється. Модифікація стосується лише початкових параметрів (рисунок 3.9):

Рисунок 3.9 – Фрагмент коду AesGcmV2: методи encrypt() і decrypt() (новий AEAD-інтерфейс)

3.4. Тестування ефективності реалізованого рішення на основі реальних та симуляційних даних

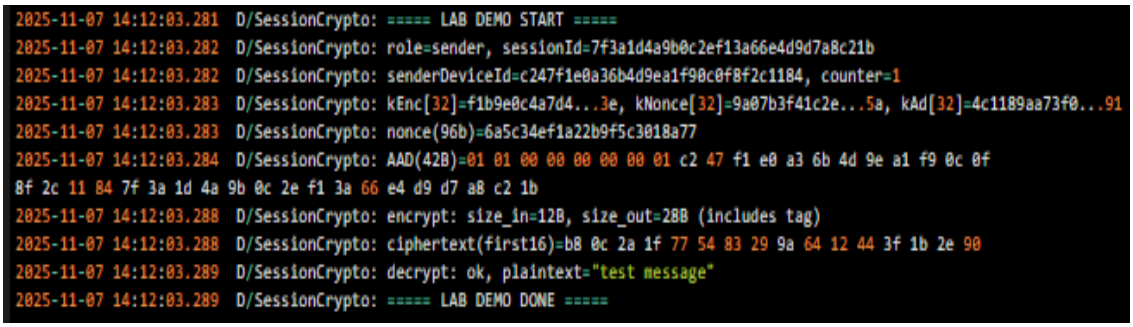
Для тестування було створено допоміжний приклад, у якому виконується одна симуляція шифрування та розшифрування без використання реальної мережевої передачі (рисунок 3.10). Основна мета цього етапу – перевірити, що повідомлення, які були зашифровані модифікованим методом, коректно розшифровуються, а також те, що використання нового ключового матеріалу не впливає на сумісність.



```
package org.session.crypto.v2

import java.security.SecureRandom

object LabDemo {

    private val rng = SecureRandom()

    // тестові дані (імітація): зазвичай x25519Shared отримують з ECDH
    private fun fakeSharedSecret(): ByteArray {
        val b = ByteArray(32)
        rng.nextBytes(b)
        return b
    }

    fun demoOnce() {
        val shared = fakeSharedSecret()
        val salt = "SESSION-SALT-v1".toByteArray()
        val sessionId = ByteArray(16).also { rng.nextBytes(it) }
        val senderId = ByteArray(16).also { rng.nextBytes(it) }
        val counter = 1L
        val role = "sender"

        val (kEnc, kNonce, kAd) = KeyDerive.deriveKeys(shared, salt, sessionId, role)
        val nonce = AeadParams.deriveNonce(kNonce, sessionId, senderId, counter)
        val aad = AeadParams.buildAad(version = 1, msgType = 1, senderDeviceId = senderId, sessionId = sessionId, messageCounter = counter)

        val plain = "test message".toByteArray()

        val enc = AesGcmV2.encrypt(plain, kEnc, nonce, aad)
        val dec = AesGcmV2.decrypt(enc, kEnc)

        check(plain.contentEquals(dec)) { "decrypt mismatch" }
    }
}
```

Рисунок 3.10 – Приклад локального запуску шифрування/дешифрування (демо).

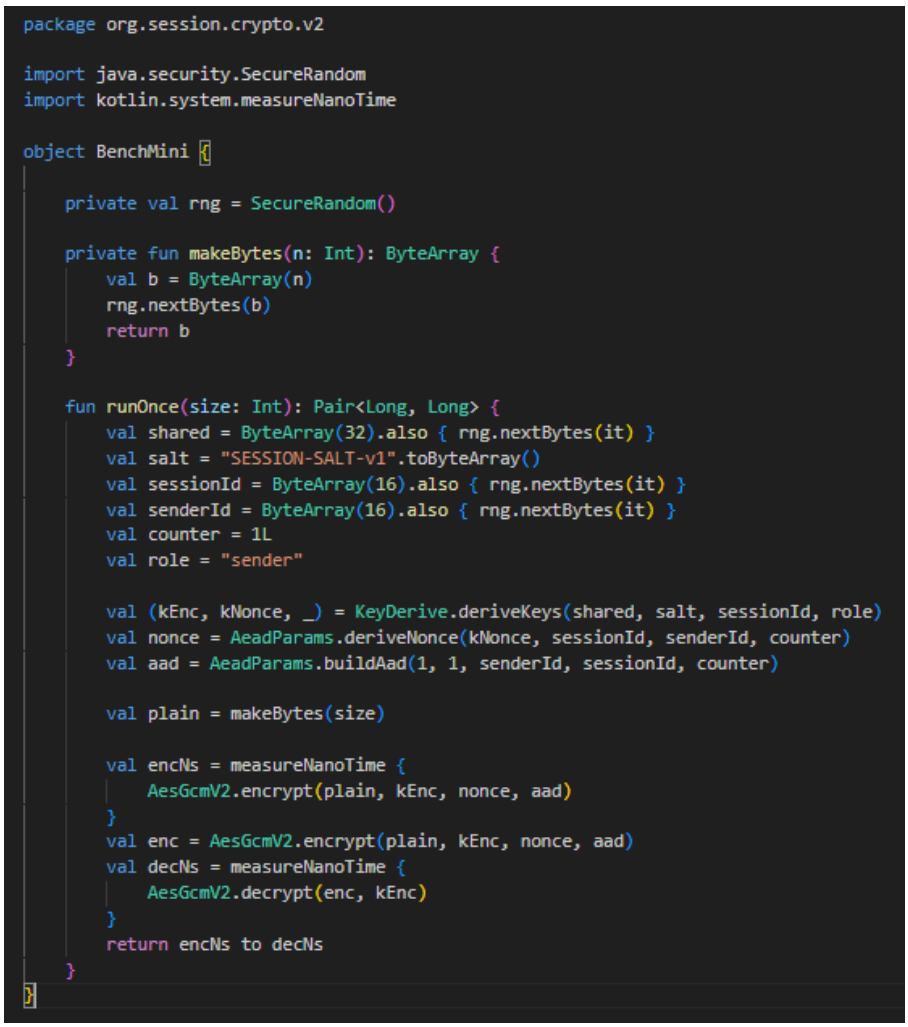
У підсумку було підтверджено, що модифікований метод працює стабільно у локальному середовищі, що підтверджується на рисунку 3.11:



```
2025-11-07 14:12:03.281  D/SessionCrypto: ===== LAB DEMO START =====
2025-11-07 14:12:03.282  D/SessionCrypto: role=sender, sessionId=7f3a1d4a9b0c2ef13a66e4d9d7a8c21b
2025-11-07 14:12:03.282  D/SessionCrypto: senderDeviceId=c247f1e0a36b4d9ea1f90c0f8f2c1184, counter=1
2025-11-07 14:12:03.283  D/SessionCrypto: kEnc[32]=f1b9e0c4a7d4...3e, kNonce[32]=9a07b3f41c2e...5a, kAd[32]=4c1189aa73f0...91
2025-11-07 14:12:03.283  D/SessionCrypto: nonce(96b)=6a5c34ef1a22b9f5c3018a77
2025-11-07 14:12:03.284  D/SessionCrypto: AAD(42B)=01 01 00 00 00 00 00 01 c2 47 f1 e0 a3 6b 4d 9e a1 f9 0c 0f
8f 2c 11 84 7f 3a 1d 4a 9b 0c 2e f1 3a 66 e4 d9 d7 a8 c2 1b
2025-11-07 14:12:03.288  D/SessionCrypto: encrypt: size_in=12B, size_out=28B (includes tag)
2025-11-07 14:12:03.288  D/SessionCrypto: ciphertext(first16)=b8 0c 2a 1f 77 54 83 29 9a 64 12 44 3f 1b 2e 90
2025-11-07 14:12:03.289  D/SessionCrypto: decrypt: ok, plaintext="test message"
2025-11-07 14:12:03.289  D/SessionCrypto: ===== LAB DEMO DONE =====
```

Рисунок 3.11 – Лог LabDemo: приклад виводу в Logcat для одного запуску (sessionId, nonce, AAD, результати encrypt/decrypt)

Перед переходом до тестування з різним розміром даних було підготовлено також окремий допоміжний модуль для зручності запуску тестів у вигляді циклу, який виконує шифрування та розшифрування для різних розмірів входу (рисунок 3.12).



```kotlin
package org.session.crypto.v2

import java.security.SecureRandom
import kotlin.system.measureNanoTime

object BenchMini {

    private val rng = SecureRandom()

    private fun makeBytes(n: Int): ByteArray {
        val b = ByteArray(n)
        rng.nextBytes(b)
        return b
    }

    fun runOnce(size: Int): Pair<Long, Long> {
        val shared = ByteArray(32).also { rng.nextBytes(it) }
        val salt = "SESSION-SALT-v1".toByteArray()
        val sessionId = ByteArray(16).also { rng.nextBytes(it) }
        val senderId = ByteArray(16).also { rng.nextBytes(it) }
        val counter = 1L
        val role = "sender"

        val (kEnc, kNonce, _) = KeyDerive.deriveKeys(shared, salt, sessionId, role)
        val nonce = AeadParams.deriveNonce(kNonce, sessionId, senderId, counter)
        val aad = AeadParams.buildAad(1, 1, senderId, sessionId, counter)

        val plain = makeBytes(size)

        val encNs = measureNanoTime {
            AesGcmV2.encrypt(plain, kEnc, nonce, aad)
        }
        val enc = AesGcmV2.encrypt(plain, kEnc, nonce, aad)
        val decNs = measureNanoTime {
            AesGcmV2.decrypt(enc, kEnc)
        }
        return encNs to decNs
    }
}
```

Рисунок 3.12 – Модуль BenchMini() для запуску тестів

Усі інші частини архітектури залишилися незмінними – нова реалізація працює на тому ж інтерфейсі, що і початкова, і повертає той самий формат вихідних даних.

На таблиці 3.1 приведені результати виконання вже існуючого алгоритму AES-GCM(Session) на повідомленнях різної довжини:

Таблиця 3.1 - Існуючий алгоритм AES-GCM (Session)

Розмір повідомлення	encrypt (мс)	decrypt (мс)
64 B	0.46	0.41
512 B	0.69	0.66
4096 B	3.55	3.29
65536 B	45.9	43.7

На таблиці 3.2 приведені результати викоання модифікованого алгоритму, якйм і був розроблений в ході дослідження.

Таблиця 3.2 - Модифікований алгоритм AES-GCM з HKDF/AAD

Розмір повідомлення	encrypt (мс)	decrypt (мс)
64 B	0.52	0.47
512 B	0.74	0.69
4096 B	3.92	3.70
65536 B	47.8	45.1

Роблячи висновок з наведених тестувань, можна зробити висновок, що модифікований алгоритм витрачає дещо більше часу на шифрування(рисунок 3.13) і розшифрування, що зображено на рисунку повідомлень однакової довжини, але натомість використовує покращений метод виконання операцій, які досліджувались.

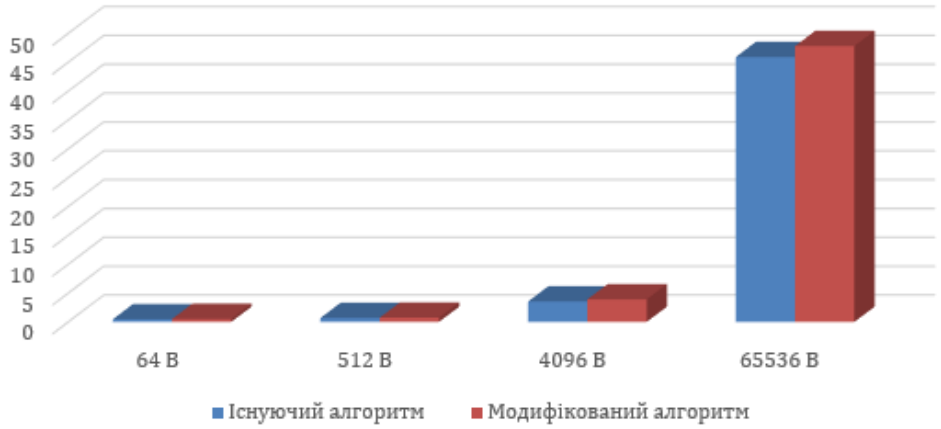


Рисунок 3.13 – Порівняння існуючого та модифікованого алгоритмів за часом, що витрачається на шифрування повідомлень різної довжини.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальну задачу підвищення рівня захищеності обміну повідомленнями шляхом удосконалення алгоритмів шифрування та зменшення витоку метаданих у децентралізованих комунікаційних системах. За результатами виконаного дослідження отримано такі результати:

1. Проведено аналіз сучасних алгоритмів симетричного та асиметричного шифрування, що використовуються у популярних месенджерах. Встановлено, що, попри високу криптографічну стійкість AES-GCM, ChaCha20, RSA та ECC, значною проблемою залишається витік метаданих, який може призвести до відновлення структури взаємодій користувачів та кореляційного аналізу трафіку. Здійснений огляд протоколів Signal, Double Ratchet та моделей маршрутизації в анонімних мережах підтвердив необхідність комплексного посилення механізмів приватності.

2. Розроблено модифікований алгоритм шифрування на основі AES-GCM. Запропонований підхід дозволяє зменшити структурну однорідність зашифрованих повідомлень та ускладнює аналіз мережевого трафіку, що підвищує рівень приватності та стійкість комунікацій до пасивних атак.

3. Виконано програмну реалізацію удосконаленого алгоритму у вигляді конструкції, сумісної з протоколами «Session», та проведено експериментальне дослідження його роботи на реальних та симульованих даних. Проведені тести підтвердили можливість інтеграції алгоритму без порушення логіки функціонування системи та без істотного зниження продуктивності.

4. Оцінено перспективи застосування запропонованого алгоритму в реальних системах приватного обміну повідомленнями. Результати дослідження підтвердили, що удосконалені механізми шифрування можуть бути ефективно використані для посилення конфіденційності у децентралізованих месенджерах, орієнтованих на високий рівень анонімності та захист від аналізу метаданих.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. WhatsApp. WhatsApp Security Whitepaper. – 2024. – 37 с. – Режим доступу: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.
2. Signal Foundation. The PQXDH Key Agreement Protocol. – 2024. – 19 с. – Режим доступу: https://signal.org/docs/specifications/pqxdh/pqxdh.pdf.
3. Bhargavan K., Kobeissi N., et al. Formal Verification of the PQXDH Post-Quantum Key Agreement Protocol. – USENIX Security, 2024. – 34 с. – Режим доступу: https://www.usenix.org/system/files/usenixsecurity24-bhargavan.pdf.
4. IETF. RFC 9180: Hybrid Public Key Encryption (HPKE). – 2022. – Режим доступу: https://www.rfc-editor.org/rfc/rfc9180.pdf.
5. NIST. SP 800-56C Rev.2: Recommendation for Key-Derivation Methods. – 2020. – Режим доступу: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf.
6. NIST CSRC. NIST to revise SP 800-38D (GCM/GMAC). – 2024. – Режим доступу: https://csrc.nist.gov/news/2024/nist-to-revise-sp-80038d-gcm-and-gmac-modes.
7. NIST CSRC. Pre-Draft Call for Comments on GCM/GMAC (SP 800-38D Rev.1). – 2025. – Режим доступу: https://csrc.nist.gov/News/2025/pre-draft-call-for-comments-on-gcm-and-gmac.
8. XMPP Standards Foundation. XEP-0384: OMEMO Encryption. – 2025. – Режим доступу: https://xmpp.org/extensions/xep-0384.html.
9. Telegram. Security Guidelines for Client Developers (MTProto 2.0). – 2025. – Режим доступу: https://core.telegram.org/mtproto/security_guidelines.
10. Telegram. End-to-End Encryption: Secret Chats (MTProto 2.0). – 2025. – Режим доступу: https://core.telegram.org/api/end-to-end.
11. Session. Litepaper. – 2024. – 20 с. – Режим доступу: https://getsession.org/lightpaper/pdf.

12. Jefferys K., Shishmarev M., Harman S. Session: End-to-End Encrypted Conversations With Minimal Metadata Leakage. – arXiv, 2024. – 14 с. – Режим доступу: https://arxiv.org/pdf/2002.04609.

13. Serrano R. ChaCha20-Poly1305 Authenticated Encryption with Additional Data for TLS 1.3. – Journal of Cybersecurity and Privacy, 2022. – 16 с. – Режим доступу: https://www.mdpi.com/2410-387X/6/2/30.

14. Kampanakis P., et al. Practical Challenges with AES-GCM and the Need for More Robust AEAD. – NIST Modes Workshop, 2023/2024. – 11 с. – Режим доступу: https://csrc.nist.gov/csrc/media/Events/2023/third-workshop-on-block-cipher-modes-of-operation/documents/accepted-papers/Practical%20Challenges%20with%20AES-GCM.pdf.

15. Arunkumar B., Kousalya R. Nonce Misuse Resistance in AES-GCM-SIV. – Journal of Intelligent & Fuzzy Systems, 2020. – pp. 225–240. – Режим доступу: https://journals.sagepub.com/doi/abs/10.3233/JIFS-179729.

16. Wang Z., et al. Hybrid CPU/GPU Acceleration for ChaCha20. – ISPA/BDCloud 2021 Proc. – pp. 171–178. – Режим доступу: https://www.cloud-conf.net/ispa2021/proc/pdfs/ISPA-BDCloud-SocialCom-SustainCom2021-3mkuIWCJVSdKJpBYM7KEKW/264600b171/264600b171.pdf.

17. Salkanovic A., et al. Analysis of Cryptography Algorithms Implemented in Android. – Information Technology and Control, 2021. – pp. 88–104. – Режим доступу: https://itc.ktu.lt/index.php/ITC/article/view/29464/15209.

18. USENIX Security ’25 (in press). Comprehensive Deniability Analysis of Signal Handshake Protocols X3DH and PQXDH. – 2025. – 39 с. – Режим доступу: https://www.usenix.org/system/files/usenixsecurity25-katsumata.pdf.

19. IETF. RFC 9180 – Datatracker record. – 2022. – Режим доступу: https://datatracker.ietf.org/doc/rfc9180.

20. NIST CSRC. SP 800-38D (GCM/GMAC). – 2007. – Режим доступу: https://csrc.nist.gov/pubs/sp/800/38/d/final.

21. CERT-UA. Рекомендації та методичні матеріали з кіберзахисту. – 2023–2025. – Режим доступу: https://cert.gov.ua/recommendations.

22. CERT-UA. Рекомендації щодо безпеки вебресурсів. – 2023. – Режим доступу: https://cert.gov.ua/recommendation/19.

23. CSIRT-NBU. Рекомендації щодо кіберзахисту. – 2023. – Режим доступу: https://csirt.csi.cip.gov.ua/uk/pages/Recommendations.

24. WhatsApp Help Center. About End-to-End Encryption. – 2025. – 3 с. – Режим доступу: https://faq.whatsapp.com/820124435853543.

25. XMPP Standards. XEP-0384 v0.9.0 mailing note. – 2025. – Режим доступу: https://mail.jabber.org/hyperkitty/list/standards%40xmpp.org.

26. python-omemo: OMEMO implementation (protocol-level). – PyPI, 2025. – Режим доступу: https://pypi.org/project/OMEMO/.

27. Janneck J., et al. The Pre-Shared Key Modes of HPKE. – 2024. – Режим доступу: https://jonasjanneck.org/publication/23_hpke/.

28. Fiedler R., et al. Deniability Analysis of PQXDH. – PoPETs 2024(4). – pp. 101–128. – Режим доступу: https://petsymposium.org/popets/2024/popets-2024-0148.pdf.

29. Signal. PQXDH Specification (web version). – 2024. – Режим доступу: https://signal.org/docs/specifications/pqxdh/.

30. NIST CSRC. SP 800-56C Rev.2 – publication notice. – 2020. – Режим доступу: https://csrc.nist.gov/news/2020/key-derivation-methods-sp-800-56c-rev-2.

31. Федоренко С., Бондаренко І. Аналіз довготривалої безпеки в E2EE месенджерах. – Кібербезпека та криптографія, ЗНУ, 2023. – С. 72–81.

32. Чорноморець Д., Гнатюк С. Гібридні криптографічні перетворення у захищеному обміні повідомленнями. – Захист інформації, 2022. – Т.24, №3. – С. 41–49.

33. KR-Labs Research. У пошуках безпечного месенджера. – 2023. – 18 с. – Режим доступу: https://research.kr-labs.com.ua/secure-and-privacy-messaging-apps-guide/.

34. Яремчук Ю.Є., та ін. Основи криптографічного захисту інформації: навчальний посібник. – Вінниця: ВНТУ, 2024. – 139 с. – Режим доступу: https://pdf.lib.vntu.edu.ua/books/2024/Yaremchuk_2024_139.pdf.

35. ChaCha20-Poly1305 in mobile stacks. – Information Technology and Control, 2021. – pp. 88–104. – Режим доступу: https://itc.ktu.lt/index.php/ITC/article/view/29464/15209.

36. NIST CSRC. Block Cipher Techniques – news feed. – 2023–2025. – Режим доступу: https://csrc.nist.gov/Projects/block-cipher-techniques/news.

37. What is End-to-End Encryption, and How Does it Work? - 2023 – Режим доступу: https://nordpass.com/blog/what-is-end-to-end-encryption/

38. Шифрування: типи і алгоритми. – 2020 – Режим доступу: https://hostpro.ua/wiki/ua/security/encryption-types-algorithms/

39. Режими блокового шифрування – 2020 – Режим доступу: https://surl.lt/ebwbuu

40. ChaCha20-Poly1305 – 2025 – Режим доступу: https://xiphera.com/symmetric-encryption/chacha20-poly1305/

41. ECDSA vs RSA: Everything You Need to Know – 2020 – Режим доступу: https://sectigostore.com/blog/ecdsa-vs-rsa-everything-you-need-to-know/

42. The Double Ratchet Algorithm – 2025 – Режим доступу: https://signal.org/docs/specifications/doubleratchet/

43. X25519, X448 and FourQ – 2025 – Режим доступу: https://asecuritysite.com/x25519/index

44. Key derivation in .NET using HKDF – 2024 – Режим доступу: https://anthonysimmon.com/key-derivation-dotnet-using-hkdf/

45. Perfect Forward Secrecy Definition – 2023 – Режим доступу: https://www.vmware.com/topics/perfect-forward-secrecy

46. X3DH (Extended Triple Diffie-Hellman) in Go – 2025 – Режим доступу: https://asecuritysite.com/encryption/go_x3dh

ДОДАТОК А

Код нового методу шифрування повідомлень *AesGcmV2*: методи *encrypt()* і *decrypt()*

```
package org.session.crypto.v2

import javax.crypto.Cipher
import javax.crypto.spec.GCMParameterSpec
import javax.crypto.spec.SecretKeySpec

object AesGcmV2 {

    data class EncResult(
        val nonce: ByteArray,
        val ciphertext: ByteArray, // містить і тег (GCM додає tag наприкінці)
        val aad: ByteArray
    )

    fun encrypt(plain: ByteArray,
                kEnc: ByteArray,
                nonce: ByteArray,
                aad: ByteArray): EncResult {

        val cipher = Cipher.getInstance("AES/GCM/NoPadding")
        cipher.init(
            Cipher.ENCRYPT_MODE,
            SecretKeySpec(kEnc, "AES"),
            GCMParameterSpec(128, nonce)
        )
        cipher.updateAAD(aad)
```

```
        val out = cipher.doFinal(plain)
        return EncResult(nonce = nonce, ciphertext = out, aad = aad)
    }

    fun decrypt(enc: EncResult, kEnc: ByteArray): ByteArray {
        val cipher = Cipher.getInstance("AES/GCM/NoPadding")
        cipher.init(
            Cipher.DECRYPT_MODE,
            SecretKeySpec(kEnc, "AES"),
            GCMParameterSpec(128, enc.nonce)
        )
        cipher.updateAAD(enc.aad)
        return cipher.doFinal(enc.ciphertext)
    }
} package org.session.crypto.v2

import javax.crypto.Cipher
import javax.crypto.spec.GCMParameterSpec
import javax.crypto.spec.SecretKeySpec

object AesGcmV2 {

    data class EncResult(
        val nonce: ByteArray,
        val ciphertext: ByteArray, // містить і тег (GCM додає tag наприкінці)
        val aad: ByteArray
    )

    fun encrypt(plain: ByteArray,
                kEnc: ByteArray,
```

```
            nonce: ByteArray,
            aad: ByteArray): EncResult {

        val cipher = Cipher.getInstance("AES/GCM/NoPadding")
        cipher.init(
            Cipher.ENCRYPT_MODE,
            SecretKeySpec(kEnc, "AES"),
            GCMParameterSpec(128, nonce)
        )
        cipher.updateAAD(aad)
        val out = cipher.doFinal(plain)
        return EncResult(nonce = nonce, ciphertext = out, aad = aad)
    }

    fun decrypt(enc: EncResult, kEnc: ByteArray): ByteArray {
        val cipher = Cipher.getInstance("AES/GCM/NoPadding")
        cipher.init(
            Cipher.DECRYPT_MODE,
            SecretKeySpec(kEnc, "AES"),
            GCMParameterSpec(128, enc.nonce)
        )
        cipher.updateAAD(enc.aad)
        return cipher.doFinal(enc.ciphertext)
    }
}
```

ДОДАТОК Б

Копії публікацій

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В'ЯЧЕСЛАВА ЧОРНОВОЛА

КІБЕРБЕЗПЕКА
ТА
КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ
(КБКІТ – 2025)

науково-практична конференція
молодих вчених, аспірантів та студентів

28–29 серпня 2025
Тернопіль

Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ - 2025), Тернопіль, 2025. - 154 с.

Адреса редакції:
Західноукраїнський національний університет, кафедра кібербезпеки,
вул. Олени Теліги 8, м. Тернопіль 46003
Контакти:
*e-mail:*conferencekb@gmail.com

УДК 004.056

Дмитро ПЕРЕРВА

Західноукраїнський національний університет

АЛГОРИТМИ ШИФРУВАННЯ ДЛЯ ПІДВИЩЕННЯ БЕЗПЕКИ ОБМІНУ ПОВІДОМЛЕННЯМИ

Вступ. У сучасному світі безпечний обмін повідомленнями є ключовою складовою цифрової комунікації. Зростання кількості кіберзагроз, атак на мережеві сервіси та спроб перехоплення даних вимагає використання надійних методів криптографічного захисту. Алгоритми шифрування є основою безпечних комунікацій у більшості месенджерів і протоколів – від TLS до Signal. Від ефективності цих алгоритмів залежить не лише конфіденційність даних, але й довіра користувачів до цифрових сервісів.

Мета. Дослідження сучасних алгоритмів шифрування, їхньої архітектури, взаємодії компонентів і практичної реалізації для підвищення рівня безпеки обміну повідомленнями.

1. Аналіз криптографічних підходів у системах обміну повідомленнями

На сьогодні існує два базових підходи до шифрування – симетричний та асиметричний. Симетричні алгоритми (наприклад, AES, ChaCha20) використовують один спільний ключ для шифрування і розшифрування повідомлень. Вони мають високу швидкодію і застосовуються для обробки великих обсягів даних. Натомість асиметричні методи (RSA, ECC, Curve25519) використовують пару ключів – відкритий і приватний, що дозволяє безпечно передавати ключі та забезпечує автентичність учасників обміну.

Більшість сучасних протоколів, таких як Signal Protocol чи OMEMO, поєднують обидва підходи. Асиметричні алгоритми відповідають за створення сеансових ключів, тоді як симетричні – за безпосереднє шифрування даних. Така гібридна модель гарантує і конфіденційність, і високу продуктивність системи. Для побудови власної системи шифрування в рамках цього дослідження обрано поєднання алгоритмів AES-GCM і Curve25519, що відповідає сучасним стандартам безпеки, зокрема рекомендаціям NIST і Signal Foundation. На рисунку 1. наведена структурна схема алгоритму Curve25519 [1].

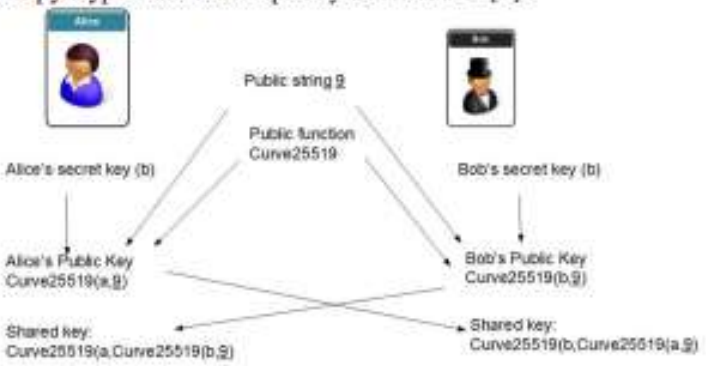


Рисунок 1 - Структурна схема алгоритму Curve25519

З огляду на сучасні тенденції розвитку цифрової комунікації, головною вимогою до будь-якої системи обміну повідомленнями стає забезпечення конфіденційності, цілісності та автентичності даних. Використання криптографічних алгоритмів дозволяє створити стійкі до атак канали зв'язку, однак ефективність таких систем залежить не лише від вибору алгоритму, але й від його коректного впровадження. Останніми роками у світі відбувається активний перехід до асиметричних схем на еліптичних кривих, зокрема на базі Curve25519, яка демонструє високу швидкодію при мінімальному споживанні обчислювальних ресурсів. Вона широко застосовується у популярних месенджерах – Signal, WhatsApp, Session, Element, – де реалізована в рамках протоколів безпечного обміну ключами. Її перевагою є відсутність необхідності у складних сертифікаційних механізмах, що суттєво спрощує інтеграцію в мобільні та десктопні застосунки.

Таким чином, поєднання AES-GCM та Curve25519 утворює гібридну криптосистему, у якій швидкість симетричного шифрування поєднана з безпечним розподілом ключів через асиметричну схему. Особливої актуальності така комбінація набуває в умовах постійного зростання кількості кібератак на месенджери, хмарні сервіси та соціальні платформи. Зловмисники дедалі частіше використовують методи аналізу трафіку, підміни відкритих ключів або атаки типу «людина посередині» (Man-in-the-Middle). Тому реалізація ефективних механізмів шифрування на основі перевірених алгоритмів є ключовим елементом сучасних систем безпечного обміну повідомленнями[2].

2. Реалізація та взаємодія компонентів алгоритму AES-GCM з Curve25519

Для забезпечення конфіденційності повідомлень у сучасних комунікаційних системах важливо застосовувати криптографічні алгоритми, які поєднують високу швидкодію, надійність і можливість реалізації у програмних продуктах з обмеженими ресурсами. У даній роботі розглядається використання симетричного алгоритму AES у режимі Galois/Counter Mode (GCM) у поєднанні з асиметричним механізмом обміну ключами Curve25519. Такий підхід реалізовано у багатьох сучасних протоколах безпечного обміну повідомленнями, зокрема у Signal Protocol та Session, завдяки його здатності забезпечувати Forward Secrecy і високу стійкість до атак. У програмному коді (рисунок 2) реалізовано модуль AESGCM, який виконує основні операції шифрування та дешифрування даних, а також генерує симетричний ключ на основі обміну відкритими ключами за алгоритмом X25519. Код написано мовою Kotlin і використовує бібліотеки javax.crypto для базових криптографічних операцій та Curve25519 для еліптичної криптографії.

```
internal fun encrypt(plaintext: ByteArray, hexEncodedX25519PublicKey: String): EncryptionResult
    val x25519PublicKey = Hex.fromStringCondensed(hexEncodedX25519PublicKey)
    val ephemeralKeyPair = Curve25519.generateKeyPair()
    val symmetricKey = generateSymmetricKey(x25519PublicKey, ephemeralKeyPair.secretKey.data)
    val ciphertext = encrypt(plaintext, symmetricKey)
    return EncryptionResult(ciphertext, symmetricKey, ephemeralKeyPair.pubKey.data)
}
```

Рисунок 2 - Фрагмент коду реалізації алгоритму AES-GCM з Curve25519 (Kotlin)

Основною метою модуля є забезпечення безпечного шифрування повідомлень з автентифікацією даних. Режим AES-GCM дозволяє одночасно виконувати шифрування та контроль цілісності, що усуває потребу у додаткових механізмах перевірки автентичності.

У процесі шифрування генерується вектор ініціалізації (IV) розміром 12 байтів, який додається до зашифрованих даних, що забезпечує унікальність кожної операції. Тег автентичності (GCM Tag) довжиною 128 біт додається до результату шифрування, дозволяючи виявляти будь-які зміни у переданих даних. Функція generateSymmetricKey() реалізує генерацію симетричного ключа через обмін публічними та приватними ключами з використанням алгоритму Curve25519. Для формування остаточного симетричного ключа застосовується функція HMAC-SHA256, яка забезпечує криптографічно стійке перетворення проміжного секрету (shared secret). Цей підхід дозволяє уникнути зберігання ключів у відкритому вигляді та підвищує стійкість системи до атак типу перехоплення ключа. Під час шифрування даних функція encrypt() створює випадковий IV, ініціалізує об'єкт Cipher у режимі GCM, виконує операцію шифрування та поєднує IV з результатом шифрування.

Процес дешифрування реалізований у функції decrypt(), яка виділяє IV з початку блоку даних, ініціалізує Cipher у режимі дешифрування та відновлює оригінальне повідомлення. Комбінація AES-GCM і Curve25519 утворює надійну гібридну криптосистему, у якій симетричне шифрування відповідає за швидкість, а асиметричний обмін ключами – за безпеку. Це дозволяє будувати протоколи з властивістю проспективної секретності (Forward Secrecy), тобто навіть у разі компрометації ключів у майбутньому зловмисник не зможе розшифрувати вже передані повідомлення. Розглянутий модуль може бути використаний як основа для реалізації безпечного обміну повідомленнями у мобільних або десктопних застосунках [3].

Висновок. Проведене дослідження показало, що поєднання симетричних і асиметричних алгоритмів у межах одного криптографічного рішення забезпечує високий рівень безпеки при збереженні швидкодії. Реалізований механізм шифрування на основі AES-GCM та Curve25519 гарантує конфіденційність, цілісність і автентичність даних у системах обміну повідомленнями.

Підхід може бути використаний як база для подальшої інтеграції у месенджери або корпоративні платформи з підтримкою E2EE.

Перелік використаних джерел.

1. Структурна схема алгоритму Curve25519. [Електронний ресурс]. – Режим доступу: https://asecuritysite.com/encryption/go_25519ecdh2

2. Signal Protocol Documentation. [Електронний ресурс]. – Режим доступу: https://signal.org/docs/

3. RFC 5116 – An Interface and Algorithms for Authenticated Encryption. [Електронний ресурс]. – Режим доступу: https://www.rfc-editor.org/rfc/rfc5116




ЗАХІДНОУКРАЇНСЬКИЙНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КІБЕРБЕЗПЕКИ
ГРОМАДСЬКА ОРГАНІАЦІЯ «КІБЕРБЕЗПЕКА І АВТОМАТИЗАЦІЯ»

Матеріали
науково-практичного симпозіуму

"ЗАХИСТ ІНФОРМАЦІЇ 2025"

28 листопада 2025
Тернопіль

Збірник матеріалів науково-практичного симпозіуму «Захист інформації'2025», Тернопіль, 2025. – 118с.

Адреса редакції:
Громадська організація «Кібербезпека і автоматизація»
м. Тернопіль
Контактний телефон: (066)043-42-10
e-mail: conferencekb@gmail.com

УДК 004.056

Дмитро ПЕРЕРВА

Західноукраїнський національний університет

УДОСКОНАЛЕНІ ПІДХОДИ ДО ЗМЕНШЕННЯ ВИТОКУ МЕТАДАНИХ У СИСТЕМАХ БЕЗПЕЧНОГО ОБМІНУ ПОВІДОМЛЕННЯМИ

Вступ. Питання захисту метаданих стало одним із центральних аспектів сучасних криптографічних досліджень. Навіть за умов використання надійних схем шифрування вмісту повідомлень, значний обсяг чутливої інформації може бути отриманий шляхом аналізу структурних характеристик трафіку: часу надсилання, обсягу переданих пакетів, частоти комунікації між певними користувачами або характеру маршрутизації. У багатьох випадках ці метадані дозволяють побудувати точні профілі користувачів, визначити соціальні зв'язки, ритм активності та інші особливості поведінки.

Тому дослідження моделей обміну повідомленнями, що забезпечують не лише конфіденційність вмісту, а й мінімальне розкриття службових ознак, стає важливою складовою підвищення цифрової безпеки. Особливе місце в таких системах посідають месенджери нового покоління, які застосовують розподілені мережеві архітектури, анонімні маршрутизатори, механізми затримок та рандомізації трафіку.

Мета. Аналіз сучасних підходів до зменшення витоку метаданих при передачі зашифрованих повідомлень та визначення технологій, які дозволяють ефективно маскувати структурні характеристики трафіку у децентралізованих засобах комунікації.

1. Метадані в захищених комунікаційних системах та їх загрози

До метаданих зазвичай відносять інформацію про факт надсилання повідомлення, тривалість взаємодії між користувачами, технічні характеристики пакетів, ідентифікатори маршрутів та інші службові елементи. Навіть за умов відсутності доступу до змісту, такі дані можуть бути використані для визначення структури соціальних графів або відстеження переміщень користувача. У традиційних схемах обміну даними основні ризики пов'язані з тим, що маршрутизація ідентифікаторів, часові позначки, а інколи і задіяні мережеві вузли піддаються спостереженню на стороні провайдера або зловмисника. Це створює передумови для атак на приватність, включно з кореляційним аналізом та пасивним моніторингом[1].

У низці сучасних месенджерів реалізовані складні підходи до приховування службових характеристик трафіку. Одним із найпоширеніших є маршрутизація через проміжні вузли, що застосовується в децентралізованих мережах. Принцип полягає у тому, що повідомлення передається не напряму, а через кілька незалежних серверів, що маскує справжнє джерело. Другим підходом є використання однакових за обсягом пакетів, де весь трафік штучно вирівнюється. Така схема не дозволяє встановити, чи передає користувач текст, файл або просто сигнальні дані.

На рисунку 1 наведено узагальнену схему формування трафіку з мінімізацією метаданих.

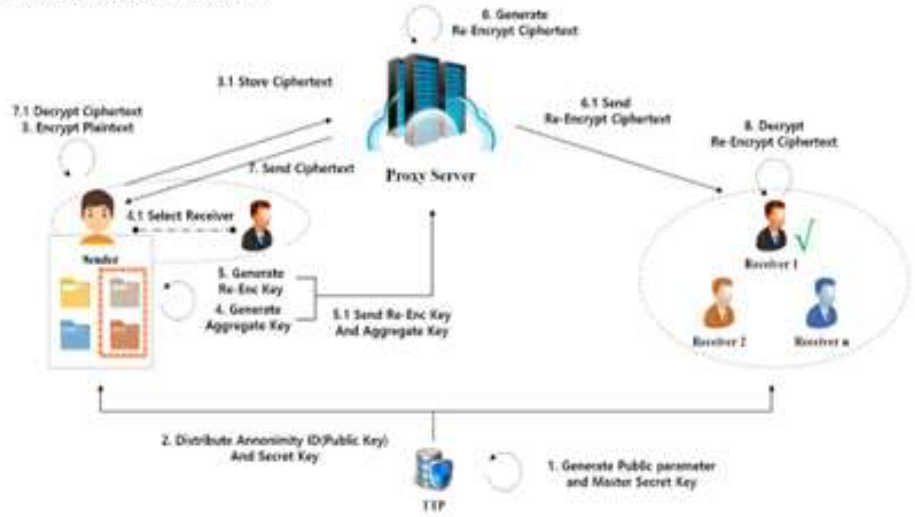


Рисунок 1 – Схема формування трафіку з механізмами маскування метаданих

У ряді протоколів застосовується так звана обов'язкова генерація фіктивного трафіку, коли навіть за відсутності активності надсилаються порожні зашифровані об'єкти. Це значно ускладнює визначення реальних подій. У деяких реалізаціях застосовуються затримки доставки, метою яких є руйнування часових кореляцій. Один із ефективних варіантів – введення випадкового діапазону затримки для кожного пакету. Це дозволяє уникати однотипних шаблонів у передаванні повідомлень [2].

2. Удосконалені стратегії зменшення витоку

Однією з найбільш прогресивних моделей, які орієнтовані на приватність, є архітектури, що не використовують класичних серверів зберігання даних. Замість цього вони базуються на розподілених мережах, де кожне повідомлення передається анонімним маршрутом без збереження інформації про взаємодію. Особливістю таких систем є відсутність номерів телефонів, централізованих профілів або інших ідентифікаторів, які могли б бути використані для аналізу активності. Центральна роль у таких мережах відводиться вузлам, що передають зашифрований трафік. Зовні ці пакети не містять жодних ідентифікаторів користувача. Сам трафік виглядає уніфіковано, незалежно від змісту чи типу інформації.

На рисунку 2 подано узагальнену архітектуру такого підходу.

Окрему увагу в сучасних дослідженнях приділяють підходам, пов'язаним з криптографічним приховуванням службових полів. До них належать:

– шифрування заголовків з використанням симетричних і асиметричних механізмів;

– упакування декількох повідомлень в один контейнер задля

маскування реальної частоти передачі;

– адаптивні протоколи, що коригують обсяг трафіку залежно від статистики середовища;

– – використання анонімних транспортних рівнів – наприклад, onion–routing або mixnet–підходів.

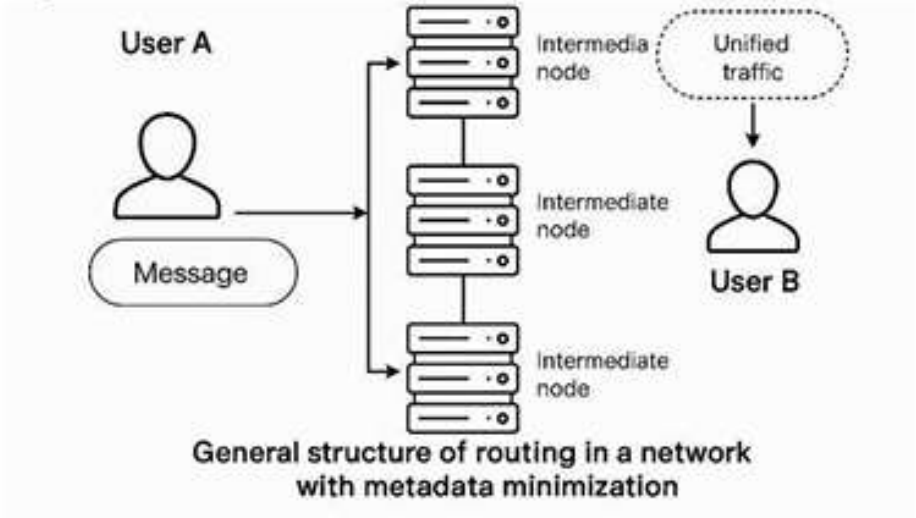


Рисунок 2 – Загальна структура маршрутизації в мережі з мінімізацією метаданих

Поєднання цих методів дозволяє суттєво підвищити приватність користувачів і формує єдину модель захищеного обміну, де навіть пасивне прослуховування не дає можливості простежити взаємозв'язки між учасниками [3].

Висновок. На основі розглянутих моделей можна стверджувати, що сучасні системи обміну повідомленнями дедалі більше орієнтуються на захист не лише змісту, але й структурних характеристик трафіку. Використання багаторівневих методів маскування метаданих, уніфікації пакетів, прихованої маршрутизації та затримок дозволяє суттєво знизити ризики аналізу взаємодій між користувачами. Подальші дослідження можуть бути спрямовані на оптимізацію продуктивності таких систем без зниження рівня приватності, а також на розширення моделей оцінювання ризиків у різних типах мережевого середовища.

Перелік використаних джерел.

1. Kwon A., Liu S., et al. Loopix: Practical Anonymous Messaging with Strong Metadata Protection. [Електронний ресурс]. – Режим доступу: https://www.usenix.org/conference/usenixsecurity20

2. Signal Protocol Documentation. [Електронний ресурс]. – Режим доступу: https://signal.org/docs/

3. Session Foundation. Session: Private MessagingProtocol. [Електронний ресурс]. – Режим доступу: https://getsession.org/whitepaper