Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

# Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**ЗВОРОТНА РОЗРОБКА ДИЗАЙНУ ПРОТОКОЛІВ ШИФРУВАННЯ**»

Виконала: студентка 4-го року навчання,
Освітньої програми «Інженерія програмного забезпечення», 121
Кропивницька Валерія Василівна
Керівник Бабич Т.А.,_____
старший викладач
Рецензент

_____
(прізвище та ініціали)

Кваліфікаційна робота захищена з оцінкою

_____
Секретар ЕК

_____
«____» _____
20____ р.

Київ – 20___

# ЗМІСТ

**ВСТУП** ..... **4**

**РОЗДІЛ 1. ОГЛЯД СТАНУ ПИТАННЯ** ..... **6**

**1.1 Загальні підходи до шифрування в месенджерах** ..... **6**

**1.2 Протокол Signal: архітектура, компоненти, принципи шифрування** .. **8**

**1.3 Застосування протоколу Signal у WhatsApp** ..... **10**

**1.4 Методи зворотної розробки в мобільній безпеці** ..... **11**

**РОЗДІЛ 2. ОГЛЯД СПЕЦИФІКАЦІЇ ПРОТОКОЛУ SIGNAL** ..... **17**

**2.1 Обмін ключами за допомогою алгоритму Extended Triple Diffie-Hellman** ..... **17**

**2.1.1 Публікація ключів на сервер** ..... **21**

**2.1.2 Встановлення спільного секрету на основі опублікованих одержувачем ключів і відправка повідомлення** ..... **22**

**2.1.3 Отримання повідомлення** ..... **24**

**2.2 Обмін зашифрованими повідомленнями з використанням протоколу Double Ratchet** ..... **25**

**2.2.1 Симетричний рачет** ..... **27**

**2.2.2 Diffie-Hellman рачет** ..... **28**

**2.2.3 Подвійний рачет** ..... **29**

**РОЗДІЛ 3. ХІД ДОСЛІДЖЕННЯ** ..... **30**

**3.1 Статичний аналіз WhatsApp** ..... **30**

**3.2 Динамічний аналіз** ..... **35**

**3.2.1 Динамічний аналіз процесу шифрування** ..... **38**

**3.2.2 Динамічний аналіз процесу дешифрування** ..... **42**

**3.2 Співставлення реалізації з специфікацією Signal протоколу** ..... **45**

**РОЗДІЛ 4. МЕТОДОЛОГІЯ ЗВОРОТНОЇ РОЗРОБКИ** ..... **47**

**РОЗДІЛ 5. ВИСНОВКИ..... 49**

**ВИКОРИСТАНІ ДЖЕРЕЛА ..... 50**

# ВСТУП

Сьогодні, коли інформаційні технології глибоко інтегровані у життєдіяльність, важко не звернути увагу на проблему інформаційної безпеки у їх використанні. Популярність мобільних додатків та глобальний обсяг переданих даних лише зростають, роблячи забезпечення конфіденційності щоденної комунікації одним із найбільш чутливих аспектів у сфері інформаційного захисту.

Перебування держави у військовому стані лише посилює критичну важливість конфіденційності та цілісності у комунікації. Цифрова безпека окрім своєї основної ролі у питанні приватності, стає ще й важливим аспектом у питанні захисту життя, координації дій як військового, так і цивільного населення.

Основні принципи інформаційної безпеки – шифрування каналів зв’язку, шифрування переданих даних, відмовостійкість апаратного та програмного забезпечення, контроль доступу, автентифікація користувачів – формують комплексний підхід до забезпечення цілісності, конфіденційності та доступності даних у цифровій комунікації.

У цій роботі сфокусована увага на шифруванні переданих даних, зокрема у месенджерах. Протоколи шифрування відіграють ключову роль — вони забезпечують захист від несанкціонованого доступу, перехоплення або модифікації повідомлень під час їх передавання.

Якась частина мобільних додатків для обміну повідомленнями має відкритий код, що дозволяє небайдужій спільноті проаналізувати аспект безпечності листування у певному застосунку. Та більшість популярних додатків мають закритий програмний код, а їх користувачі можуть судити про безпеку своїх повідомлень лише судячи з заяв авторів цих застосунків про те, який підхід вони використовують у забезпеченні приватності листувань.

Одним із найбільш популярних та поширених додатків є WhatsApp, який заявляє, що використовує Signal Protocol у шифруванні повідомлень. Надійність Signal протоколу підтверджується численними аудитами та аналізами, однак

попри відкритість специфікацій Signal, факт його реалізації у WhatsApp та інших додатках з закритим кодом, залишається недоступним для прямого аналізу, і тому недослідженим з практичного боку. Саме тому виникає потреба у зворотній розробці – техніці, що дозволяє без відкритого коду досліджувати внутрішню архітектуру, логіку взаємодії компонентів, та інші властивості програмних продуктів у виконуваних файлах.

Предметом дослідження у цій роботі є методи зворотної розробки для ідентифікації реалізації протоколів шифрування у мобільних додатках, а також аналіз того, як саме реалізовані окремі криптографічні операції, обмін ключами, механізми шифрування та дешифрування.

Метою дослідження є виявлення внутрішньої реалізації протоколу шифрування повідомлень у мобільному застосунку WhatsApp на базі операційної системи iOS, а також опис цієї реалізації, її аналіз та співставлення з офіційними специфікаціями протоколу Signal.

# РОЗДІЛ 1. ОГЛЯД СТАНУ ПИТАННЯ

## 1.1 Загальні підходи до шифрування в месенджерах

Аспект безпеки є особливо важливим у мобільних додатках, які мають функцію листування. Тим більше, враховуючи той факт, що вони використовуються мільйонами користувачів щодня. Передача повідомлень, файлів, голосові дзвінки повинні бути надійно захищені від несанкціонованого доступу та компрометації листування. На сьогодні існує низка підходів, які формують поточні стандарти індустрії інформаційної безпеки передачі даних та напрямки її розвитку.

Основним підходом до забезпечення безпеки комунікації у мережі є використання TLS протоколу, який полягає у шифруванні даних, що передаються, на транспортному рівні. Даний протокол спрямований на безпеку клієнт-серверних з'єднань, захищаючи від атак класу man-in-the-middle. Втім, у цьому випадку сервер має доступ до незашифрованих даних, що не відповідає принципам повної конфіденційності.

Покращенням є використання наскрізного шифрування (англ. *end-to-end encryption*), яке полягає у гарантії того, що лише відправник і отримувач повідомлення можуть розшифрувати його вміст. До того ж, шифрування повинно відбуватись на стороні відправника, а не на сервері, адже сервер також не повинен мати доступ до вмісту даних, що передаються. Це досягається за допомогою криптографії з відкритим ключем, а також принципах встановлення спільних секретів на початку обміну повідомленнями і виведення набору ключів для шифрування та дешифрування повідомлень на основі цих спільних секретів.

Підміна ключів - це атака, можлива у наскрізному шифруванні, якщо не реалізовані або порушені механізми перевірки автентичності ключів. Додаток, що реалізує такий тип шифрування, повинен приділяти особливу увагу автентифікації користувачів і їх ключів. Зокрема, найпростішим способом може бути повідомлення через інтерфейс, що ключ контакту змінився. Також

можливим варіантом може бути порівняння ключових відбитків вручну за допомогою спеціальних кодів у вигляді цифр або QR-кодів, що можуть бути згенеровані додатком.

Окрім забезпечення безпечної передачі даних у мережі, важливо також адресувати проблему безпечного зберігання локальних даних, якщо вони присутні. Локальний кеш повинен зберігатись лише у зашифрованому вигляді. Особлива увага повинна приділятись збереженню ключів локально. При наявності альтернатив зберіганню кеша на диску – захищених середовищах зберігання даних на пристрої, перевага надається їм. Наприклад, на мобільних пристроях на базі операційної системи iOS наявний Keychain, який спроектований саме для безпечного зберігання ключів, токенів та інших надчутливих даних.

Отже, сьогодні стандартами індустрії у забезпеченні приватності комунікації у месенджерах є використання TLS протоколу на транспортному рівні для передачі даних між клієнтом та сервером, наскрізне шифрування з механізмом перевірки автентичності ключів для унеможливлення несанкціонованого доступу до повідомлень та зашифроване зберігання локально на пристрої закешованих даних та ключів.

## 1.2 Протокол Signal: архітектура, компоненти, принципи шифрування

Вдалі підходи до забезпечення конфіденційності та цілісності передачі даних стають об’єктами різноманітних стандартів та протоколів. Сьогодні одним із найпопулярніших та найпрогресивніших протоколів наскрізного шифрування є протокол Signal.

Протокол Signal – це сучасний криптографічний протокол, що був створений компанією Open Whisper Systems для забезпечення наскрізного шифрування повідомлень, голосових викликів та відео викликів. Мета протоколу – гарантувати, що лише відправник і отримувач можуть читати передані дані, навіть якщо ці дані зберігаються на сервері або у разі проведення man-in-the-middle атаки, тобто якщо третя сторона слухає прослуховує канал зв’язку. Сьогодні цей протокол застосовується в багатьох популярних додатках, таких як однойменний Signal, WhatsApp, Facebook Messenger та інших.

Архітектура протоколу забезпечує високу стійкість до атак і забезпечує цю стійкість навіть у випадках, коли окремі ключі скомпрометовано. Протокол Signal поєднує у своїй архітектурі два основних компоненти. Першим основним компонентом є обмін ключами X3DH (Extended Triple Diffie-Hellman), який проходить у кілька етапів, для асинхронного встановлення сесії між двома користувачами навіть у разі, якщо один із них не в мережі. Другим основним компонентом є використання Double Ratchet для шифрування повідомлень, де кожне нове повідомлення шифрується унікальним ключем, який оновлюється після кожної передачі, тим самим забезпечуючи пряму секретність (англ. *forward secrecy*) – властивість криптографічних систем зберігати конфіденційність минулих ключів сесії при компрометації довготривалого ключа.

Використовується також концепція початкового ключа (англ. *prekey*) – тимчасовий початковий публічний ключ, який користувач завантажує на сервер заздалегідь, щоб інші могли почати з ним сесію, навіть якщо він не онлайн. Зокрема тут і реалізовується принцип асинхронності. На сервері зберігається не лише один початковий ключ, а їх набір. У цей набір, зокрема, входить

підписаний приватним ключем публічний початковий ключ та набір одноразових початкових ключів (англ. *one-time prekey*). Користувач, який хоче надіслати перше повідомлення іншому користувачу, бере набір початкових ключів цього іншого користувача з серверу – підписаний початковий ключ і один із одноразових ключів. Після цього він шифрує ними своє ініціююче повідомлення, і відправляє. Отримувач отримає його і одразу зможе відповісти, бо з цього повідомлення сформується спільний секрет. Таким чином, початкові ключі не лише забезпечують асинхронність, а і забезпечують пряму секретність від самого початку передачі даних.

У якості криптографічних примітивів використовуються Curve25519, AES-256-CBC або AES-256-GCM (залежно від реалізації), та HMAC-SHA256.

Шифрування повідомлень здійснюється за допомогою використання одного із вищезгаданих протоколів AES,  ключ шифрування для кожного нового повідомлення в результаті оновлення стану сесії, коли генерується новий симетричний ключ. До повідомлення додається HMAC-підпис для перевірки цілісності повідомлення. Одержувач використовує свою локальну копію сесії, щоб обчислити ключ для дешифрування і перевірити підпис.

Таким чином комунікація є асинхронною, адже усі сторони використовують свої локальні незалежні копії стану конкретної сесії. Також протокол передбачає механізм перевірки автентичності ключів. Користувач повинен мати змогу звірити з іншою стороною комунікації короткий код безпеки, який генерується в будь-який момент на основі стану сесії.

Таким чином, протокол Signal реалізує усі сучасні стандарти безпеки мережевої комунікації і є відмінним вибором для впровадження у месенджері.

### 1.3 Застосування протоколу Signal у WhatsApp

Додаток WhatsApp, що належить компанії Meta, є одним із найпопулярніших месенджерів. Компанією було заявлено, що Signal протокол реалізований у ньому.

Загалом функціонал додатку WhatsApp дозволяє зареєструватись у ньому за допомогою номеру мобільного телефону. Після реєстрації можна заходити у акаунт із різних пристроїв, тобто у дизайні шифрування повинна бути присутня мультидевайсна архітектура.

Після початку чату із якимось користувачем, візуальний інтерфейс повідомляє, що повідомлення будуть захищені наскрізним шифруванням. У меню контакту в чаті можна зайти у підменю, яке стосується шифрування та безпеки чату. Там згенерований QR-код і окремо код, що складається з цифр. Якщо ці два елементи однакові у обох сторін чату, тоді сеанс вважається захищеним. QR-код можна відсканувати один в одного або відправити цифровий код якимось іншим каналом зв’язку для порівняння.

У візуальному інтерфейсі все вказує на те, що у даному застосунку реалізований Signal протокол, проте WhatsApp – це застосунок із закритим код, що унеможливлює прямий аудит відповідності реалізації стандартам протоколу. Аналіз реалізації можливий лише з використанням методів зворотної розробки, які дозволяють вивчати додаток як статично, так і динамічно. Це дозволяє виявити окремі компоненти коду – класи, методи, об’єкти, виявити їхні функції та поведінку та співставити це все зі специфікацією протоколу.

## 1.4 Методи зворотної розробки в мобільній безпеці

Зворотна розробка (англ. *reverse engineering*) – це процес аналізу готового продукту (апаратного чи програмного забезпечення) з метою вивчення його будови, функціональності, поведінки, алгоритмів чи протоколів його роботи, без доступу до початкового коду чи документації. Зворотна розробка включає в себе різні підходи до дослідження виконуваного файлу як у форматі його знаходження на диску, так і у форматі запущеного процесу, коли виконуваний файл завантажений у пам’ять. Поєднання цих підходів дозволяє отримати комплексне уявлення про внутрішні механізми застосунку.

Дослідження виконуваного файлу у форматі, коли він просто знаходиться на диску, називається статичним аналізом. У ході статичного аналізу реверс інженер дивиться на залежності, імпортовані та експортовані символи, рядки, проініціалізовані глобальні та статичні дані, функції, локальні типи, тощо. Цей першочерговий огляд дозволяє реверс інженеру проаналізувати виконуваний файл на предмет дослідження, наприклад, знайти релевантні рядки чи функції. Таким чином реверс інженер підтверджує наявність елементів, цікавих у рамках дослідження, позначає для себе важливі та пріоритетні місця у виконуваному файлі, які далі будуть досліджуватись ґрунтовніше поєднанням методів статичного та динамічного аналізу.

Існує багато інструментів для статичного аналізу. Зазвичай використовується інструментарій на основі дизасемблера та декомпілятора, інструменти для аналізу структури та сегментів виконуваного файлу, програмне забезпечення для розпаковування, програмне забезпечення для деобфускації, програмне забезпечення для редагування секцій з релокаціями, тощо.

У даному дослідженні використовувався інструмент IDA – інструментарій на основі дизасемблера для виконуваних файлів, який дозволяє проводити статичний аналіз бінарних файлів, виявляти функції, посилання на символи, об’єкти класів, аналізувати ARM64 інструкції у мобільних iOS застосунках,

будувати та досліджувати графи виконання та керування, дивитись посилання на використання функцій, рядків та інших конструкцій коду.

Динамічний аналіз спрямований на аналіз виконуваного файлу у форматі запущеного процесу. Цей тип досліджень є більш гнучким та інформативним, адже дозволяє обходити різноманітні обфускації та дивитись що насправді відбувається з застосунком, що він виконує, що підвантажує у свій адресний простір. У ході динамічного аналізу реверс інженер зв'язує частини коду, які мають динамічне обрахування адрес переходів, аналізує пам'ять, відновлює структури, аналізує call stack функцій, зв'язує дані і структури з функціями, що їх викликають. Це дозволяє досліднику отримати повну комплексну картину про функціональність застосунку, порядок виконання функцій, локальні типи даних та вплив цього процесу на стан системи загалом.

Для динамічного аналізу використовуються інструменти відлагодження програм для того, щоб розставляти брейкпоінти і аналізувати виконання конкретних місць у коді, переглядати регістри, стек, пам'ять, і виконання коду покроково. Для мобільних застосунків на iOS основним відлагоджувачем є нативний lldb, а також debugserver, який дозволяє проводити віддалені сеанси відлагодження процесу застосунку. З допомогою debugserver і проксі-інструментів, як от іроху, можна відлагоджувати застосунки безпосередньо на пристрої з macOS, використовуючи з'єднання через USB.

Також важливою групою інструментів для динамічного аналізу є інструменти для впровадження скриптів у процес, зокрема для трасування функцій, встановлення хуків на методи, перехоплення викликів API. На iOS основним таким інструментом є Frida та Frida-trace. Frida – це динамічний інструментарій для впровадження JavaScript-коду у процеси, що дозволяє підключатися до запущених процесів. Група інструментів Frida дозволяє модифікувати поведінку програм у реальному часі, отримувати зміщення методів у виконуваному файлі для подальшого використання методів статичного аналізу, взаємодіяти із системними викликами libSystem для аналізу впливу процесу на систему. У контексті реверс інженерії iOS додатків важливим є

можливість перехоплювати та логувати Objective-C методи, виводити всі Objective-C класи, які доступні у запущеному процесі, що є особливо важливим при дослідженнях в умовах обфускації або просто при великій кількості виконуваних файлів, що завантажені у процес.

Важливим елементом динамічного аналізу є також інструменти для моніторингу файлової системи, які дозволяють перехоплювати низькорівневі операції над файлами, такі як виклики open(2), read(2), write(2), fcntl(2), fstat(2), та інші. Це дозволяє досліджувати наявність закешованих даних на пристрої, конфігураційних файлів, та інших важливих елементів роботи процесу.

Не можна не згадати й інструменти для перехоплення, моніторингу та аналізу мережевого трафіку, що дозволяє дослідити залежність процесу застосунку від мережі, що застосунок відправляє, що отримує, тощо. Найпопулярнішими аналізаторами трафіку є Fiddler, mitmproxy, Burp Suite, Charles, Wireshark. Wireshark є особливо корисним для аналізу внутрішніх незадокументованих протоколів, оскільки перехоплює трафік різних протоколів, не лише HTTP/HTTPS.

У більшості практичних застосувань реверс інженерії ефективним підходом є поєднання інструментів статичного та динамічного підходів. У контексті аналізу роботи криптографічних протоколів у закритих системах поєднання статичного та динамічного аналізу дозволяє виявити внутрішню логіку реалізації, подивитись на виклики функцій, локалізувати функції з обробкою та ротацією ключів, подивитись на те, який вигляд мають ці ключі, дослідити процес шифрування та дешифрування даних.

## 1.5 Огляд структури рантайму в мові програмування Objective-C

Objective-C – це мова із динамічною типізацією, яка підтримує об’єктно-орієнтовану парадигму програмування. Мова програмування Objective-C задумувалась як надбудова над мовою C, і її синтаксичні конструкції спрямовані на керування мовою C. Саме бібліотека часу виконання, Objective-C runtime library, приносить мові її особливості, реалізуючи можливості динамічної типізації і забезпечуюючи можливості об’єктно-орієнтованого програмування. Objective-C runtime library скомпонована у всі застосунки, написані мовою Objective-C. Сама ж реалізація бібліотеки окрема для кожної платформи, на iOS бібліотека називається libobjc.A.dylib.

Дана мова має низку як переваг, так і недоліків, проте основне, що цікавить нас у контексті реверс-інженерії – це властивість цієї мови зберігати широкий набір метаданих про свої мовні конструкції. Для реверс інженера це є надзвичайною перевагою, адже це дозволяє комформтно досліджувати структуру застосунку динамічно, навіть за умов обфускації або відсутності символів.

Усі об’єкти під час виконання програми представлені у вигляді звичайної C-структури *objc_object.* Кожен об’єкт має вказівник на об’єкт свого класу – isa-вказівник.

```
typedef struct objc_class *Class;
typedef struct objc_object {
    Class isa;
} *id;
```

Класи Objective-C під час виконання представлені структурами objc_class. Структури objc_class і objc_object майже ідентичні між собою. Об’єкт класу також тримає isa-вказівник на свій метаклас. Метакласи – це спеціальні об’єкти класів, які описують поведінку класових методів.

```
struct objc_class {
    Class isa;
};
```

Однією з ключових особливостей Objective-C, яка робить процес реверс-інженерії дуже зручним – це відкладене зв’язування методів, об’єктів та повідомлень. У статичних мовах програмування зв’язування методів виконується на етапі компіляції коду. А у застосунках, написаних на мові Objective-C, рішення про те, який метод викликати приймається безпосередньо під час виконання програми. Коли об’єкту надсилається повідомлення, бібліотека часу виконання приймає це рішення на основі селекторів та таблиць диспетчеризації. Конкретніше, виклик метода відбувається за допомогою виклику *objc_msgSend*. Така модель дозволяє переглядати усі викликані функції динамічно, навіть якщо виклик цієї функції не видно у дизасембльованому коді. Також це дозволяє використовувати такі техніки як method swizzling, динамічні додавання методів до класу, а також зручно перехоплювати виклики, що є також неабияк корисним у процесі динамічного аналізу та реверс-інженерії загалом.

Селектори – це конструкція мови Objective-C, задачею якої є ідентифікування методів. Селектор має ім’я методу, проте він не є конкретною функцією чи прямим вказівником на його реалізацію. Селектори лежать у основі концепції відкладеного зв’язування методів, а саме реалізації системи динамічного надсилання повідомлень до об’єктів у мові, адже коли у об’єкта викликається певний метод, то бібліотека часу виконання не викликає цей метод напряму, а шукає його через селектор. Тобто коли ми викликаємо у коді метод, наприклад –[MyClass myMethod], то насправді викликається objc_msgSend(self, @selector(myMethod)).

```
typedef struct objc_selector       *SEL;
```

У контексті реверс-інженерії наявність селекторів у пам’яті, або у відповідних секціях виконуваного Mach-O файлу, таких як __objc_selrefs та __objc_methname дозволяє легко знаходити назви методів та швидко

локалізовувати ключові точки взаємодії між об’єктами різних класів. Збереження таких метаданих про назви класів і методів відкривають легкий шлях до перехоплення викликів функцій, а централізовані виклики всіх конструкцій об’єктно-орієнтованої парадигми через функцію objc_msgSend, дозволяють легко трасувати конкретні функції конкретних класів з допомогою таких інструментів, як Frida-trace.

Інформація про конкретну імплементацію методів класу зберігається в структурі IMP. У системі бібліотеки часу виконання кожен метод асоціюється з селектором і відповідною імплементацією. Ці зв’язки зберігаються в таблицях диспетчеризації методів, які прив’язані до класів. Ці таблиці можна відновити або дослідити навіть у випадку часткового видалення символів. Під час виконання, коли об’єкт отримує повідомлення на виклик методу, то виконується пошук у відповідному класі методу, який відповідає певному селектору (SEL), і виклик його імплементації (IMP).

Таким чином розуміння внутрішнього влаштування механізмів часу виконання мови Objective-C є критично важливим при проведенні реверс-інженерії застосунків на платформі iOS, оскільки є великий шанс, що частина кодової бази застосунку написана саме на Objective-C. Розуміючи засади цієї мови, можна підходити до процесу реверс-інженерії з багатьох сторін. Використовуючи інструмент Frida, можна через API ObjC.classes, ObjC.selector та інші, отримувати доступ до структури класів та їхніх методів динамічно. Використовуючи інструмент IDA, можна аналізувати selRef-и, classRef-и, та інші конструкції дизасембльованого коду, щоб якісно проаналізувати структуру виконуваного файлу.

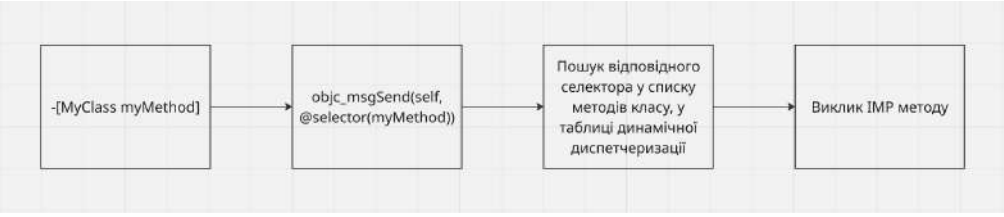


*Рисунок 1. Загальна схема відкладеного зв’язування методів*

# РОЗДІЛ 2. ОГЛЯД СПЕЦИФІКАЦІЇ ПРОТОКОЛУ SIGNAL

## 2.1 Обмін ключами за допомогою алгоритму Extended Triple Diffie-Hellman

Однією із ключових складових Signal протоколу є обмін ключами за алгоритмом Extended Triple Diffie-Hellman (X3DH). Метою обміну ключів є взаємна автентифікація обох сторін обміну і встановлення спільного секрету між ними як підтвердження пройденої взаємної перевірки. Основними перевагами X3DH є асинхронність за дизайном, пряма секретність та криптографічна стійкість.

У якості криптографічних примітивів за протоколом X3DH використовуються еліптичні криві X25519 або X448, хеш функції SHA-256 або SHA-512. Застосунок, який використовує Signal протокол, повинен визначити типи примітивів, які він буде використовувати, та також встановити якесь певне Info – ASCII рядок, який буде ідентифікувати цей застосунок і використовуватись у функції деривації ключів (англ. *KDF, Key Derivation Function*).

У контексті WhatsApp використання X3DH є вдалим саме через асинхронність, оскільки це не передбачає обов’язкове знаходження в мережі обох сторін обміну. Усі користувачі публікують свої публічні ключі на сервер. Ті, хто хочуть надіслати якомусь користувачу повідомлення, беруть їх опублікований публічний ключ на сервері і використовують його, щоб зашифрувати повідомлення і встановити спільний секрет. Користувач, який отримує таке зашифроване повідомлення, розшифрує його і також обрахує цей спільний секрет.

Таким чином, у X3DH є три основні ролі:

- користувач А (відправник) який в конкретний момент хоче безпечно надіслати повідомлення, маючи можливість в будь-який момент зашифрувати його так, щоб його розшифрував лише одержувач
- користувач Б (одержувач), який хоче отримувати повідомлення незалежно від того, чи він зараз у мережі, хоче отримувати такі повідомлення, щоб їх міг розшифрувати лише він сам
- сервер, який виступає допоміжною ланкою в забезпеченні асинхронності комунікації між користувачем А та користувачем Б

У протоколі X3DH використовується низка криптографічних ключів, згенерованих з використанням еліптичних кривих у вигляді пар публічного і приватного ключів. Типи таких ключів у протоколі X3DH:

- Identity Key – ключ ідентичності
- Ephemeral Key – тимчасовий ключ
- Signed Prekey – підписаний початковий ключ
- One-time Prekey – одноразовий початковий ключ

Усі ключі, як вже згадано вище, повинні бути або у формі X25519 або X448, залежно від обраних параметрів у імплементації застосунку.

Identity Key (ключ ідентичності) – це довготривалий ключ, який має кожен користувач у системі. У криптографічній нотації Extended Triple Diffie-Hellman позначається як IK. У контексті ролей у X3DH (одержувач, відправник) ключ ідентичності однаково важливий як для ролі одержувача, так і для ролі відправника.

Ephemeral Key (тимчасовий ключ) – це короткотривалий ключ, який генерується при кожній ітерації обміну ключів. У криптографічній нотації Extended Triple Diffie-Hellman позначається як EK. У контексті ролей, цей ключ використовується відправником. При кожній ітерації обміну ключів використовується унікальна пара ключа ідентичності і тимчасового ключа, де ключ ідентичності сталий.

Signed Prekey (підписаний, заздалегідь підготовлений початковий ключ) – це підписаний ключ з існуванням середньої тривалості, який він публікує на сервер. У криптографічній нотації Extended Triple Diffie-Hellman позначається як SPK. У контексті ролей, цей ключ в основному важливий для одержувача, оскільки дозволяє відправнику встановлювати з ним комунікацію.

One-time prekey (одноразовий початковий ключ) – ключ одноразового використання, який також опублікований на сервер. Користувач публікує не лише один одноразовий ключ, а їх набір.

Якщо на сервері закінчаться одноразові ключі, то встановлення сесії комунікації з цим користувачем буде відбуватися із меншим рівнем безпеки, тому потрібно, щоб на сервері завжди була достатня їх кількість. Проте, дивно, що цей ключ є опціональним. У специфікації протоколу його роль не є критичною. Якщо він наявний на сервері, то шифрування має один «лишній» крок, а якщо одноразові ключі  закінчились – цей крок пропускається.

У криптографічній нотації Extended Triple Diffie-Hellman позначається як OPK. У контексті ролей, цей ключ також використовується одержувачем зі схожою метою, як і Signed Prekey, проте реалізує додатковий шар безпеки для встановлення унікальної сесії саме з цим користувачем.

Такі ключі не передаються під час самої ініціалізації спільного секрету або, іншими словами, сесії, а вже знаходяться на сервері у момент, коли інший користувач хоче почати захищену комунікацію. Таким чином ці два типи ключів є ключовими у забезпеченні асинхронної роботи протоколу. Пара ключів Signed Prekey та One-time prekey для ролі одержувача виконують ту ж саму роль, що і Ephemeral Key для відправника. Signed prekey забезпечує, що це ключ саме того користувача, якому буде призначене повідомлення, а one-time prekey забезпечує унікальність сесії саме з цим користувачем.

У разі успішного обміну ключами, між користувачами встановлюється спільний секрет, який можна вважати ще одним типом ключів у архітектурі X3DH. У криптографічній нотацій позначаємо його SK, що відповідає назві

Secret Key. Такий ключ має довжину 32 байти і буде використовуватись для подальшого шифрування переданих даних у сесії між користувачами.

Окрему увагу хочеться приділити ще раз опціональності одноразового ключа (one-time prekey), яку можна назвати недоліком протоколу Extended Triple Diffie-Hellman, адже це призводить до можливості проведення атак повторного відтворення. Атака повторного відтворення може призводити до формування однакового Secret Key у різних сесіях, що без додаткової подальшої рандомізації ключа шифрування, може призводити до порушення конфіденційності передачі даних.

Обмін ключами за протоколом Extended Triple Diffie-Hellman можна розділити на три фази:

1. Користувач Б (одержувач) публікує свій ключ ідентичності (IK) і набір початкових ключів (SPK, OPK) на сервер.
2. Користувач А (відправник) отримує набір початкових ключів (англ. *prekey bundle*) із серверу та використовує його, щоб зашифрувати і надіслати ініціююче повідомлення для користувача Б (одержувача)
3. Одержувач отримує і опрацьовує ініціююче повідомлення, отримуючи з нього спільний секрет (SK)

### 2.1.1 Публікація ключів на сервер

У першому етапі публікується Identity Key IK, Signed Prekey SPK, окремо підпис початкового ключа SPK, та набір одноразових початкових ключів ($OPK_1$, $OPK_2$, $OPK_3$, …, $OPK_N$). Ротації Identity Key не відбувається – він довготривалий і сталий. Ротація ключів в першу чергу відбувається у наборі одноразових початкових ключів (вони оновлюються, коли їх стає мало). Також час від часу оновлюються підписаний початковий ключ та, відповідно, його підпис. Інтервал ротації ключів залежить від імплементації.

Ротація ключів відбувається не лише на сервері. Для забезпечення прямої секретності ротація також повинна відбуватися локально, адже крім ротації публічних ключів, повинно щось відбуватися і з приватними ключами. Після оновлення підписаного початкового ключа (SPK), приватний його відповідник повинен бути видалений. Приватні ключі, які відповідають опублікованим публічним одноразовим початковим ключам, повинні видалятись одразу після отримання ініціюючого повідомлення на стороні одержувача.

### 2.1.2 Встановлення спільного секрету на основі опублікованих одержувачем ключів і відправка повідомлення

Для відправки ініціюючого повідомлення відправник отримує повний набір ключів, опублікований одержувачем, із сервера. Спочатку відправник перевіряє підпис початкового ключа. На цьому етапі, якщо перевірка пройдена не успішно – обмін припиняється з помилкою. Якщо ж перевірка підпису успішна – відбувається перехід до наступних кроків протоколу.

DH(PK1, PK2) – функція обрахування спільного ключа Diffie-Hellman на основі двох публічних ключів PK1 та PK2, створених на еліптичній кривій. Саме ця функція бере участь у формуванні ініціюючого повідомлення. У подальших нотаціях одержувач буде позначатись субскриптом 1, а отримувач – субскриптом 2.

Сценарій початкового обміну ключами різний у залежності від наявності одноразових ключів отримувача на сервері. Якщо одноразові початкові ключі наявні, то обрахування спільного секрету відбувається таким чином:

$DH1 = DH(IK_1, SPK_2)$

$DH2 = DH(EK_1, IK_2)$

$DH3 = DH(EK_1, SPK_2)$

$DH4 = DH(EK_1, OPK_2)$

SK = KDF (DH1 || DH2 || DH3 || DH4), де || - це конкатенація двох ключів, а KDF – це *key derivation function,* функція деривації ключів

Якщо одноразових початкових ключів немає на сервері, то обрахування спільного секрету пропускає формування ключа DH4:

$DH1 = DH(IK_1, SPK_2)$

$DH2 = DH(EK_1, IK_2)$

$DH3 = DH(EK_1, SPK_2)$

SK = KDF (DH1 || DH2 || DH3), де || - це конкатенація двох ключів, а KDF – це *key derivation function,* функція деривації ключів

У даному обміні ключами DH1 та DH2 відповідають за проходження взаємної автентифікації сторін обміну, а DH3 та DH4 відповідають за пряму секретність.

Після проходження перевірки відбувається ротація ключів на сервері та на стороні відправника. На сервері видаляється одноразовий початковий ключ, якщо він був наявний та використовувався при обміні. На стороні відправника видаляється приватний тимчасовий ключ (Ephemeral Key, EK). Також видаляються виводи функцій DH, які брали участь у формуванні спільного секрету Secret Key (SK).

Важливим елементом повідомлення є блок асоційованих даних (англ. *AD, associated data*). Відправник формує цей блок із двох публічних ключів ідентичності. Ці публічні ключі повинні бути перетворені у послідовність певною функцією Encode, яка повинна бути визначена додатком. Блок асоційованих даних є конкатенацією перетворених функцією Encode ключів.

$$AD = Encode(IK_1) \,||\, Encode(IK_2)$$

Наступним кроком є формування та надсилання власне самого ініціюючого повідомлення. Ініціююче повідомлення включає в себе Identity Key відправника, Ephemeral Key відправника, ідентифікатори ключів одержувача, які брали участь в обміні ключами Diffie-Hellman, та тіло повідомлення, зашифроване спільним секретом, одержаним у попередні етапи, з використанням асоційованих даних.

### 2.1.3 Отримання повідомлення

Після отримання ініціюючого повідомлення, отримувач дістає ключ ідентичності (IK) та тимчасовий ключ (EK) відправника, дістає свої приватні ключі, які відповідають тим, що використовувались відправником для проведення обміну, а саме:

- приватний ключ до підписаного початкового ключа
- приватний ключ до одноразового початкового ключа

Використовуючи ці значення, отримувач обраховує ключ тими ж самим кроками, що й відправник, щоб отримати свій секрет. Важливим кроком є видалення усіх обрахованих значеннях на проміжних кроках. Також одержувач обраховує асоційовані дані із свого ключа та ключа відправника.

Наступним кроком є безпосередньо спроби розшифрувати повідомлення використовуючи свій секрет і асоційовані дані. Якщо спроба дешифрування не успішна, то обмін припиняється з помилкою.

У разі успішного обміну протокол вважається пройденим, отримувач видаляє приватний одноразовий початковий ключ для забезпечення прямої секретності.

## 2.2 Обмін зашифрованими повідомленнями з використанням протоколу Double Ratchet

Алгоритм Double Ratchet (дослівно – подвійний «рачет», тобто механізм поступового та необоротного оновлення ключів) використовується для обміну зашифрованими повідомленнями на основі спільного секретного ключа. Досить часто використовується на основі протоколу обміну ключами Diffie-Hellman. Зокрема, у випадку WhatsApp використовується саме такий стек протоколів.

Алгоритм має низку переваг, зокрема пряму секретність. Сторони обміну обраховують нові ключі для кожного повідомлення задля того, щоб старі ключі не могли бути обрахованими на основі нових.

В основі архітектури протоколу лежить поняття KDF-ланцюжків, де KDF – це key derivation function, тобто функція деривації ключа. KDF – це певна визначена додатком криптографічна функція, яка приймає параметром спільний для сторін обміну секретний ключ, випадковий KDF ключ та тіло повідомлення. Зашифровані дані є результатом роботи такої функції.

Ланцюжком дана концепція називається тому, що новим випадковим KDF ключем виступає частина виводу із попереднього кроку. Інша частина виводу із попереднього кроку виступає як вихідний ключ.

KDF-ланцюжки мають низку сильних сторін. Основними позитивними властивостями KDF-ланцюжка є:

- гнучкість – без знання KDF ключів не можна обрахувати виведені ключі
- пряма секретність – якщо третя особа в певний момент часу дізнається KDF ключ, вона не може обрахувати минулі виведені ключі
- відмовостійкість криптографічної системи

У Double Ratchet сесії між користувачами А та Б зберігається три KDF-ланцюжка:

1. Головний ланцюжок
2. Ланцюжок, що отримує або ланцюжок-отримувач
3. Ланцюжок, що відправляє або ланцюжок-відправник

У системі Double Ratchet існує два «рачети»:

1. Симетричний рачет
2. Diffie-Hellman рачет

### 2.2.1 Симетричний рачет

Симетричний рачет – це механізм оновлення ключів для безпосереднього шифрування і дешифрування повідомлень. Кожне повідомлення зашифроване унікальним ключем message key. Ці унікальні ключі виводяться із KDF-ланцюжків відправлення і отримання. При цьому симетричному рачету не властива відмовостійкість, оскільки вхідні параметри для ланцюжка-отримувача та ланцюжка-відправника є сталими. Якщо третя сторона отримує доступ до ключів якогось з ланцюжків, то вона може обрахувати всі майбутні ключі для шифрування повідомлень і таким чином дешифрувати усі майбутні повідомлення.

У симетричному рачеті ротація ключів може відбуватись після кожного повідомлення. Ключ шифрування може видалятись одразу, оскільки він більше не буде використовуватись. Проте, оскільки на основі цих ключів не обраховуються інші ключі, їх збереження ще певний інтервал часу після роботи з конкретним повідомленням може мати позитивний вплив на систему, оскільки це забезпечує правильну обробку виняткових ситуацій з повідомленнями, де якось порушується їх доставка, тощо.

### 2.2.2 Diffie-Hellman рачет

Diffie-Hellman рачет – це механізм оновлення KDF-ключів для ланцюжків, що отримують і відправляють. Під час обміну повідомленнями користувачі також обмінюються новими Diffie-Hellman ключами, а новий спільний секрет, який утворюється в результаті обміну цих ключів, стає новим вхідним параметром для головного ланцюжка. Головний ланцюжок обраховує нові виведені ключі, які стають новими KDF ключами для двох інших ланцюжків, що отримують і відправляють.

Щоб імплементувати рачет Diffie-Hellman, кожна сторона обміну генерує пару публічного та приватного ключів Diffie-Hellman, яка стає їхньою активною парою на даному кроці рачету. Поточний публічний ключ відправника включається у заголовок повідомлення, що буде передаватись. Одержувач, який отримує новий публічний ключ, також оновлює свою пару ключів у себе локально. Таке почергове оновлення пари Diffie-Hellman ключів відбувається упродовж всього обміну даними.

Почерговість оновлення ключів також забезпечує відмовостійкість криптографічної системи у разі компрометації одного з поточних приватних ключів. Скомпрометований ключ заміниться новим, який недоступний третій стороні, і тому третя сторона не зможе обрахувати правильні подальші деривації ключів на основі виводу Diffie-Hellman обміну.

### 2.2.3 Подвійний рачет

Криптографічна система, що складається з поєднаних рачетів типу Diffie-Hellman та симетричних рачетів для відправок та отримання повідомлень називається подвійним рачетом. Така система вирішує недоліки, які були б наявні при використанні кожної з концепцій окремо.

Коли отримується повідомлення із новим публічним ключем у заголовку, Diffie-Hellman рачет виконує свій крок роботи на рутовому KDF-ланцюжку. На цьому кроці оновлюються ключі для KDF-ланцюжків для отримання та відправки повідомлень.

Коли повідомлення надіслане або отримане, симетричний рачет виконує свій крок роботи на відповідному KDF-ланцюжку вже з оновленими ключами задля отримання ключа шифрування чи дешифрування повідомлення.

# РОЗДІЛ 3. ХІД ДОСЛІДЖЕННЯ

## 3.1 Статичний аналіз WhatsApp

Статичний аналіз – це один із ключових підходів до реверс-інженерії застосунків, який полягає у аналізі виконуваного файлу без його запуску, тобто у форматі того, коли він просто лежить на диску. Цей підхід дозволяє виконати початковий огляд структури застосунку, виявити ознаки його ключових функцій, дослідити його локальні типи, функції, рядки, ініціалізовані дані тощо.

У даному дослідженні WhatsApp, початковий статичний аналіз використовувався задля ідентифікації місць у виконуваному файлі, які є релевантними до імплементації Signal протоколу.

Для того, аби провести дослідження потрібно було перевести мобільний пристрій iOS у стан jailbreak – програмне рішення для обходу більшості безпекових обмежень операційної системи iOS, яке полягає у патчингу ядра при його завантаженні. Під час завантаження пристрою спочатку виконується код завантажувача (англ. *bootloader*), задача якого – взяти ядро операційної системи із безпечного сховища на пристрої і передати йому виконання. Jailbreak базується на експлуатації вразливостей у SecureROM задля виконання вищезгаданого патчингу. Таким чином, після переведення пристрою у стан jailbreak, ми отримуємо низку можливостей, без яких зворотна розробка є майже неможливою, а саме:

- відсутність sandbox
- відсутність перевірок підписів застосунків
- як наслідок, можливість встановлювати на девайс сторонні додатки
- можливість запускати інструменти відлагодження
- наявність повного доступу до всієї файлової системи
- доступ до зашифрованих або захищених регіонів пам’яті процесу застосунку
- тощо

У цьому дослідженні використовувався пристрій iPhone 6s з операційною системою iOS 15.8.3, з джейлбрейком palera1n на базі checkmate експлойта. Саме цей джейлбрейк був обраний тому, що він найстабільніше працює з цією моделлю телефона та цією версією iOS, також він підтримує rootful і rootless підходи, а альтернативний розповсюджувач застосунків дозволяє зручно завантажувати усі необхідні утиліти, такі як ssh, lldb, debugserver, Frida-server, та інші.

Застосунки на iOS розповсюджуються у форматі IPA-файлів. За своєю суттю IPA файл – це архів у форматі ZIP, який містить усі виконувані та конфігураційні файли для роботи застосунку. IPA-файли зашифровані, тому їх потрібно витягувати з пристрою під час їх виконання.

Важливими елементами IPA-файлу є:

- виконувані файли – файли, зазвичай без розширення, у форматі Mach-O
- Info.plist – конфігураційний файл, який містить метадані про застосунок, його ідентифікатори, дозволи, тощо;
- Frameworks/ - директорія, що містить додаткові фреймворки, які використовуються застосунком;
- ресурси та інші файли для коректної роботи додатку

Найбільшу цікавість для нас становлять саме виконувані файли. Для початку дослідження було вилучено з пристрою розкриптований IPA і його виконувані файли за допомогою інструменту на основі Frida, який запускає застосунок, під’єднується до процесу цього застосунку і копіює з його адресного простору всі розшифровані бінарні файли.

У ході дослідження з процесу застосунку WhatsApp було вилучено такі виконувані файли:

- WhatsApp – 64-бітний Mach-O файл з архітектурою arm64
- SharedModules – 64-бітний Mach-O файл shared library з архітектурою arm64
- WABloksKit – 64-бітний Mach-O файл shared library з архітектурою arm64

Після цього ці виконувані файли можна аналізувати, використовуючи звичні інструменти, зокрема IDA Pro, яка є одним із найпопулярніших інструментаріїв, в основі яких лежить дизасемблер. У ході статичного аналізу WhatsApp спочатку було проведено початковий огляд кожного із вищезгаданих файлів. Після завантаження, зчитавши усі секції, IDA Pro побудувала списки функцій, локальних типів, тощо. Наступним кроком було перевірити рядки на предмет наявності ключових слів таких як session, signal, key, handshake, та інших. У файлі WhatsApp таких ключових слів було не знайдено у рядках, а ось у файлі SharedModules – їх виявилась наявна велика кількість, тому далі у ході дослідження більш сконцентрувалась на саме цьому файлі.

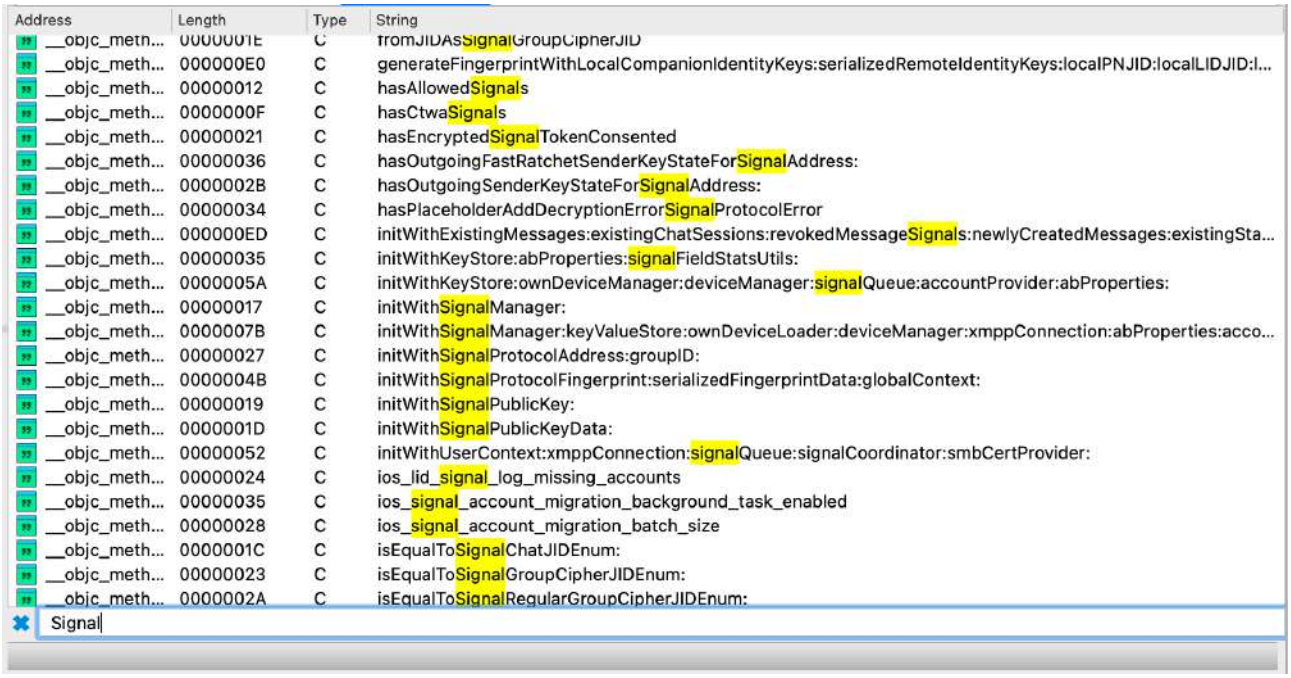


*Рисунок 2. Результати пошуку рядків за ключовим словом "Signal" у файлі SharedModules*

Результатом пошуку стала не лише локалізація потрібного файлу, а й також важливий висновок про те, що релевантна для дослідження частина коду написана на Objective-C, що значно спрощує нам задачу.

Після цього я прийняла рішення проаналізувати список класів, а саме результати пошуку аналогічно за ключовим словом Signal.


*Рисунок 3. Результати пошуку визначених локальних типів за ключовим словом "Signal" у файлі SharedModules*

Такі знайдені результати варто якось упорядковувати, щоб потім використовувати їх у наступних етапах дослідження і загалом зручно по них навігуватись. Я проаналізувала список класів, які мають префікс WASignal у своїй назві і попередньо погрупувала їх за такими ознаками, які можна припустити судячи з їх назв:

- приймає участь у дешифруванні
- приймає участь у шифруванні
- приймає участь у ініціалізації
- приймає участь у ротації ключів

| № ID | Aa Name | Category |
|---|---|---|
| 2 | WASignalCoordinator | Encryption Decryption |
| 7 | WASignalDecryptJournalId | Decryption |
| 8 | WASignalDecryptParsedMessage | Decryption |
| 26 | WASignalDecryptProtobufMessageParser | Decryption |
| 9 | WASignalDecryptRequest | Decryption Not relevant |
| 29 | WASignalEncryptProtobufMessage | Encryption |
| 27 | WASignalEncryptProtobufMessageCreator | Encryption |
| 10 | WASignalEncryptResult | Encryption |
| 28 | WASignalFieldStatsUtilsProvider | Other |
| 11 | WASignalFingerprint | Other |
| 23 | WASignalGroupCipherJIDEnum | Other Group messaging |
| 19 | WASignalInMemoryKeyStore | Other |
| 4 | WASignalKeyBundle | Key management |
| 21 | WASignalKeyStore | Key management |
| 17 | WASignalKeyStore.WAMDAccount | Key management |
| 18 | WASignalKeyStore.WAMDDevice | Key management |
| 5 | WASignalLocalKeyBundle | Key management |
| 30 | WASignalManager | Decryption Encryption |
| 13 | WASignalManagerGroupEncryptionRequest | Encryption Group messaging |
| 14 | WASignalManagerIdentityKeyFetcher | Key management |
| 12 | WASignalManagerIndividualEncryptionRequest | Encryption |
| 1 | WASignalManagerMain | Encryption Decryption |

*Рисунок 4. Згруповані класи*

Наступним кроком у дослідженні буде динамічний аналіз кожної групи цих класів.

Отже, задачею статичного аналізу є локалізація релевантних місць у виконуваному файлі для того, аби надалі провести детальніший їх аналіз вже динамічно. У ході статичного аналізу WhatsApp було виокремлено групу класів Objective-C, які містять префікс “WASignal”, та містять релевантні до суті Signal протоколу назви методів. Було зроблено припущення на основі назв класів про те, в яких концептуальних частинах протоколу вони беруть участь і згруповано їх за ознаками відігравання ролі у шифруванні повідомлень, дешифруванні повідомлень, ініціалізації стану, ротації ключів. Надалі прийнято рішення перейти до динамічного аналізу для того, щоб підтвердити чи спростувати припущення про ролі цих класів, дослідити детальніше їх функціональність та збільшити повноту уявлення про архітектуру Signal протокол у додатку.

### 3.2 Динамічний аналіз

Динамічний аналіз використовується для подальшого детального дослідження місць у коді застосунку. Такий підхід у реверс інженерії дає детальнішу та повнішу картину, адже дозволяє досліджувати поведінку застосунку під час його виконання.

У випадку з WhatsApp, динамічний підхід до дослідження стає критично важливим для виявлення логіки шифрування, передачі ключів і викликів внутрішніх методів для Signal протоколу, тому що з отриманого набору класів важко виокремити якусь вхідну точку, з якої можна покроково статично аналізувати код.

У ході динамічного дослідження WhatsApp основним інструментом , який використувався, була Frida – інструментарій на основі JS-рушія, який дозволяє впроваджуватись у процес шляхом впровадження JavaScript-коду в нативні застосунки, контролювати пам’ять їх процесів, хукати функції, та багато іншого.

У випадку з WhatsApp особливо корисними функціями Frida були можливості:

- перехоплювати виклики Objective-C методів
- виводити інформацію про список Objective-C класів
- виводити інформацію про список Objective-C методів конкретного класу
- виводити стек трейс під час виконання функцій
- логувати послідовність дій застосунку під час криптографічних операцій

Для того, аби приєднатися до процесу WhatsApp через Frida, використовувалась команда:

*frida -U -f net.whatsapp.WhatsApp -l <script name>*

Опція -U вказує на те, що ми хочемо під’єднатися до frida серверу на пристрої, що під’єднаний по USB.

Опція -l вказує на назву скрипта, який ми хочемо впровадити у цільовий процес.

Опція -f дозволяє запустити процес з вже впровадженими туди JS-скриптами і таким чином виконувати раннє трасування застосунку. У випадку з WhatsApp це дозволяє нам відловити ранні виклики методів класів, які відповідають за ініціалізацію стану застосунку, ініціалізацію стану Signal протоколу.

Щоб отримати список доступних Objective-C класів під час виконання, використовувався скрипт:

```
console.log(Object.keys(ObjC.classes).join('\n'));
```

Дослідження отриманого списку Objective-C класів під час виконання було необхідним задля виключення можливої обфускації або видалення символів з певних секцій виконуваного файлу. Якби щось було видалено або обфусковано, ми б не побачили можливий важливий набір класів і не дослідили їх.

У випадку з WhatsApp не було виявлено додаткових релевантних класів, тобто можна сказати що частина коду, написана на Objective-C, не є обфускованою.

Щоб отримати список доступних Objective-C методів класу WASignalCoordinator під час виконання, використовувався скрипт:

```
var cls = ObjC.classes.WASignalCoordinator;
console.log(cls.$methods.join('\n'));
```

Щоб перехопити виклик певного методу, тобто захукати його, використовувався скрипт:

```
Interceptor.attach(ObjC.classes.WASignalKeyBundle["-
signedPrekey"].implementation, {
  onEnter: function (args) {
    console.log("signedPrekey called");
  },
  onLeave: function (retval) {
    console.log("Returned: " + retval);
  }
});
```

Проте зручнішим є не впровадження окремих скриптів на хукання окремих функцій, а використання інструменту frida-trace, який вже ставить хуки на вказаний набір функцій та дозволяє зручно протрасувати їх виклики.

Для цього використовувалась наступна команда:

```
frida-trace -U -n WhatsApp -m "-[WASignal*]"
```

Це дозволяє логувати всі методи класів, що починаються на WASignal. Трасування всіх класів, що починаються на WASignal, виявилось неефективним, тому що зациклило вивід трасування через специфіку архітектури класів та велику кількість методів, які виступають геттерами для полів об’єктів.

Було прийнято рішення трасувати класи групами. Зокрема, було протрасовано окремо групу класів, що відповідають за шифрування, групу класів, що відповідають за дешифрування.

### 3.2.1 Динамічний аналіз процесу шифрування

У процесі шифрування окрему цікавість при трасуванні становили наступні класи:

- WASignalManager
- WASignalManagerMain
- WASignalCoordinator
- WASignalEncryptProtobufMessage
- WASignalEncryptProtobufMessageCreator
- WASignalEncryptResult
- WASignalManagerIndividualEncryptionRequest

Повна frida команда, яка використовувалась при трасуванні процесу шифрування:

```
frida-trace -U -f net.whatsapp.WhatsApp \
  -m '-[WASignalManager *]' \
  -m '-[WASignalCoordinator *]' \
  -M '-[WASignalCoordinator stateless_signal_store_context]' \
  -M '-[WASignalCoordinator signal_store_context]' \
  -M '-[WASignalCoordinator signal_global_context]' \
-m '-[WASignalEncryptProtobufMessage *]' \
-m '-[WASignalEncryptProtobufMessageCreator *]' \
-m '-[WASignalEncryptResult *]' \
-m '-[WASignalManagerIndividualEncryptionRequest *]' \
-m '-[WASignalManagerPrekeyFetcher *]' \
-m '-[WASignalManagerIndividualEncryptionRequest *]' \
-m '-[WASignalManagerIdentityKeyFetcher *]' \
-m '-[WASignalManagerPrekeyUploader *]'
```

*Рисунок 5. Фрагмент результату трасування процесу шифрування за допомогою frida-trace №1*

*Рисунок 6. Фрагмент результату трасування процесу шифрування за допомогою frida-trace №2*

Загалом результат при трасуванні вийшов доволі об’ємний, проте у ньому можна виділити декілька основних викликів, які чітко окреслюють весь процес шифрування повідомлення у WhatsApp.

Спочатку відбувається виклик –[WASignalManager encryptMessage:inChat:forDeviceJIDs:useDeprecatedSessionsOrSenderKeys:icdcMetadataByUserJID:recipientUserJIDIfIndividualChat:retryCount:], тут приходить внутрішній запит до об’єкту WASignalManager на шифрування повідомлення.

Після цього викликається –[WASignalManagerIndividualEncryptionRequest initWithMessage:chatJID:deviceJID:retryCount:useDeprecatedSession:]. Тут поглиблюється попередній виклик, виконання передається відповідальному класу і об’єкту.

Наступним важливим кроком є виклик –[WASignalManager encryptIndividualRequest:signalCoordinator:result:error], де вже бачимо передавання об’єкту signalCoordinator, який ми побачили під час статичного аналізу таким, що координує всю внутрішню низькорівневу криптографічну логіку, пов’язану із Signal протоколом.

І нарешті, найцікавішим кроком, є виклики –[WASignalCoordinator encryptPlaintextData:forSignalAddress:fastRatchet:maxPkcs7PaddingLength:ciphertextData:ciphertextType: error:] та –[ WASignalCoordinator encryptForIndividualAddress:plaintextData:ciphertextData:ciphertextType:].

Якщо перейти у останню функцію в IDA Pro, то бачимо виклики session_cipher_create, session_cipher_encrypt. Такі ж функції з такими назвами існують в офіційній бібліотеці Signal, написаній на C.


*Рисунок 7. Фрагмент дизасембльованої функції -[WASignalCoordinator encryptForIndividualAddress:plaintextData:cipher]*

### 3.2.2 Динамічний аналіз процесу дешифрування

У процесі дешифрування окрему цікавість при трасуванні становили наступні класи:

- WASignalManager
- WASignalManagerMain
- WASignalCoordinator
- WASignalDecryptParsedMessage
- WASignalDecryptProtobufMessageParser
- WASignalDecryptJournalId

Повна frida команда, яка використовувалась при трасуванні процесу дешифрування:

```
frida-trace -U -f net.whatsapp.WhatsApp \
-m '-[WASignalManager *]' \
-m '-[WASignalDecryptJournalId *]' \
-m '-[WASignalCoordinator *]' \
-M '-[WASignalCoordinator stateless_signal_store_context]' \
-M '-[WASignalCoordinator signal_store_context]' \
-M '-[WASignalCoordinator signal_global_context]' \
-m '-[WASignalDecryptParsedMessage *]' \
-m '-[WASignalDecryptProtobufMessageParser *]'
```

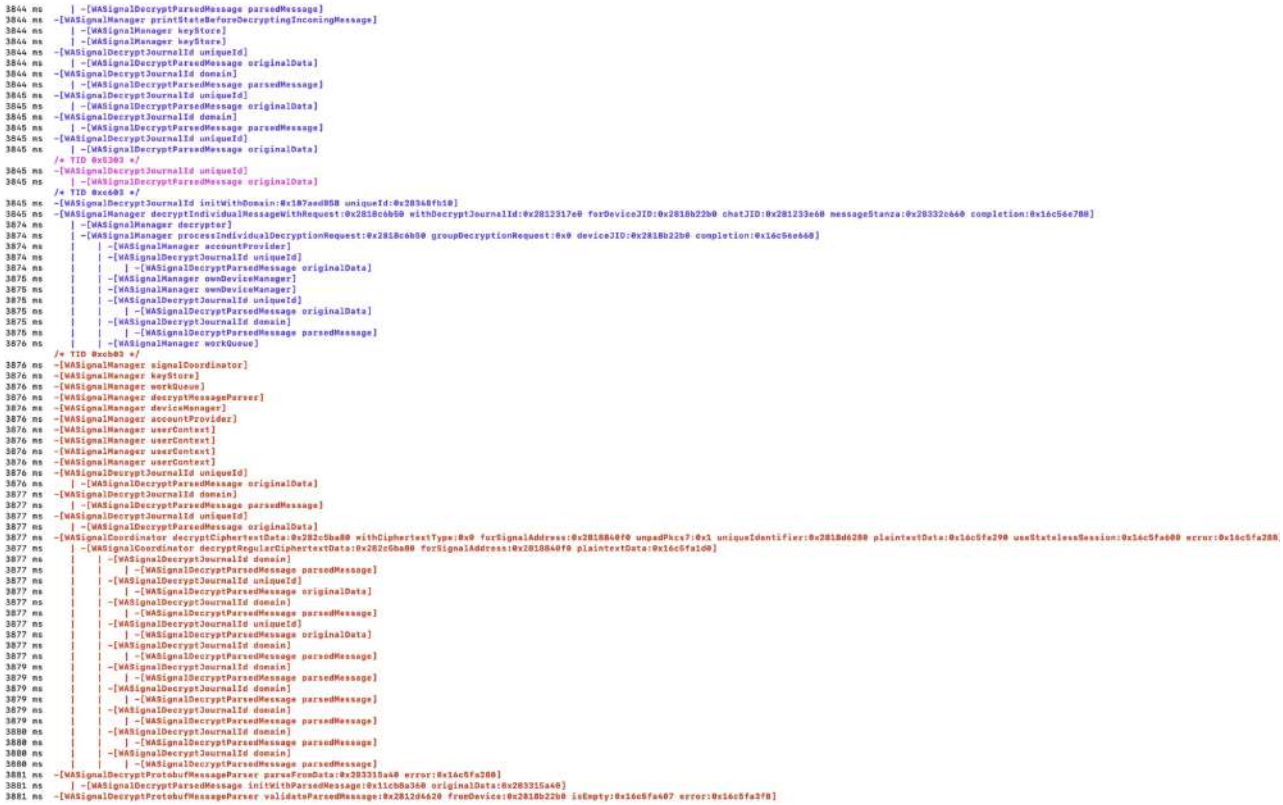

*Рисунок 8. Фрагмент трасування процесу дешифрування за допомогою frida-trace*

Результат трасування процесу дешифрування також вийшов доволі об’ємний. З цього результату було виокремлено основні кроки, які окреслюють весь процес дешифрування повідомлення у WhatsApp.

Спочатку відбувається виклик –[WASignalManager decryptIndividualMessageWithRequest:withDecryptJournalId:forDeviceJID:chatJID: messageStanza:completion:]. Тут приходить внутрішній запит до об’єкту WASignalManager на дешифрування повідомлення.

Наступним важливим викликом є –[WASignalCoordinator decryptCiphertextData:withCiphertextType:forSignalAddress:unpadPkcs7:uniqueIde ntifier:plaintextData:useStatelessSession:error:]. Тут відбувається перетворення зашифрованого тексту у звичайний. Це метод класу WASignalCoordinator, класу, який відповідає за координування бібліотекою libSignal. У цій функції всередині відбувається розгалуження викликів. Якщо це prekey повідомлення – тоді викликається функція –[WASignalCoordinator decryptPreKeyCiphertextData: forSignalAddress:plaintextData:statelessly:]. Якщо це звичайне повідомлення – тоді викликається функція –[WASignalCoordinator decryptRegularCiphertextData :

forSignalAddress : plaintextData:]. Також відбувається опрацювання випадків дешифрування типів повідомлення, властивим груповим чатам у Signal.

-[WASignalCoordinator decryptRegularCiphertextData : forSignalAddress : plaintextData:] є основною функцією для дешифрування повідомлення. Спочатку там викликається функція libSignal signal_message_deserialize для того, щоб виокремити зашифрований текст. Після цього викликається функція session_cipher_create, і, нарешті, session_cipher_decrypt_signal_message.

Наступними основними кроками у процесі дешифрування є формування повідомлення згідно з внутрішньою структурою представлення повідомлень у WhatsApp з використанням технології protobuf.

## 3.2 Співставлення реалізації з специфікацією Signal протоколу

Після проведення статичного та динамічного аналізу було виконано співставлення основних концептуальних елементів специфікації Signal протоколу з дослідженою внутрішньою реалізацією WhatsApp. Важливо врахувати, що WhatsApp ймовірно використовує офіційну бібліотеку з реалізованим Signal протоколом, написаною на C, та інтегрує її у власну архітектуру через обгортки, класи, що забезпечують взаємодію з криптографічними операціями на вищому рівні абстракції.

У WhatsApp присутні абстракції над типами ключів, які беруть участь у обміні Extended Triple Diffie-Hellman (X3DH):

- *Identity Key* – у WhatsApp зберігається у внутрішньому key store, ініціалізується під час ініціалізації стану WASignalManager. Зберігається у об'єкті класу WASignalKeyBundle, за доступ до нього відповідає метод –[WASignalKeyBundle identityPublicKeyData]
- *Signed Prekey* – зберігається у об'єкті класу WASignalKeyBundle, за доступ до нього відповідає метод –[WASignalKeyBundle signedPrekey], за доступ до його підпису відповідає метод –[WASignalKeyBundle signedPrekeySignatureData]
- *One-time Prekey* – об'єкти класу WASignalPrekeyBundle зберігають у собі ephemeralPrekey

За ініціалізацію сесії відповідає метод -[WASignalCoordinator processPreKeyBundle:forSignalAddress:error:]. Тут встановлюється спільний секрет на основі prekey bundle одержувача. Викликається session_builder_process_pre_key_bundle з бібліотеки libsignal.

Обмін повідомленнями на основі double ratchet також реалізований у WhatsApp. Симетричний рачет використовується у методах класу WASignalCoordinator, які відповідають за шифрування і дешифрування повідомлень. Внутрішньо ці методи викликають функції

session_cipher_decrypt_signal_message та session_cipher_encrypt, який працює із внутрішнім симетричним рачетом сесії.

Diffie-Hellman рачет працює внутрішньо під час дешифрування та шифрування повідомлень. Це реалізовано в бібліотеці libsignal при деривації нових ключів для ланцюжка шифрування повідомлень.

Після кожного повідомлення ключі з попередніх виводів видаляються з внутрішнього key store. Зокрема, при трасуванні виявлені виклики методів видалення ключів класу WASignalManager. Це підтверджує наявність прямої секретності у реалізації WhatsApp.

Отже, ключові компоненти протоколу мають чітке відповідне представлення у внутрішній реалізації WhatsApp. Спосіб побудови реалізації включає використання C-бібліотеки з обгортками, які інкапсулюють логіку в Objective-C класах та забезпечують масштабовану інтеграцію в архітектуру застосунку. Об’єкти класів WASignalManager, WASignalCoordinator та WASignalKeyStore виступають як обгортки для C API, де знаходиться вся низькорівнева криптографічна логіка протоколу Signal.

# РОЗДІЛ 4. МЕТОДОЛОГІЯ ЗВОРОТНОЇ РОЗРОБКИ

У результаті дослідження було не лише проаналізовано відповідність реалізації WhatsApp з специфікацією Signal протоколу, а і сформовано практичну методологію зворотної розробки дизайну протоколів шифрування у мобільних додатках із закритим кодом на платформі iOS. Методологія полягає у використанні комбінованого підходу, що поєднує статичний і динамічний аналіз виконуваних файлів.

Загально окреслити послідовність методології можна таким чином:

1. Підготовчий етап
2. Статичний аналіз
3. Динамічний аналіз
4. Змішаний аналіз
5. Співставлення з специфікацією протоколу, якщо вона наявна

У підготовчому етапі відбувається встановлення цільового застосунку на пристрій, переведення пристрою у стан jailbreak, вилучення розшифрованих виконуваних файлів з процесу та решту файлів з IPA. Також шукається інформація з відкритих джерел про те, які протоколи там можливо реалізовані, таким чином окреслюється напрям дослідження.

При статичному аналізі виконується дизасемблювання виконуваних файлів, пошук релевантних конструкцій у коді, побудова гіпотез щодо ролей класів і методів.

При динамічному аналізі виконується трасування методів за допомогою frida чи frida-trace, перехоплюються та досліджуються виклики функцій, пов’язані з шифруванням та дешифруванням. Для підтвердження та детальнішого дослідження ротації ключів пишуться хуки на методи, в яких ці ключі приймають участь, логуючи їх значення на кожній ітерації протоколу. Для детальнішого дослідження реалізації протоколу логуються і інші методи важливих функцій, аби визначити їх ролі у процесі роботи протоколу.

Останнім етапом є співставлення з специфікацією. Визначивши усі ключові конструкції у коді, вони співставляються із ключовими концептуальними поняттями у протоколі, аналізується відповідність криптографічних операцій описаній роботі криптографічної системи цільового протоколу.

Таким чином, дослідження довзолило не лише підтвердити використання протоколу Signal у WhatsApp, а й деталізувати, як саме реалізовані ключові етапи його роботи, зокрема ініціалізація та ротація ключів, встановлення сесії, обмін повідомленнями і підтримка прямої секретності. Запропонована методологія може бути використана для дослідження внутрішніх реалізацій інших протоколів у мобільних застосунках з закритим кодом.

# РОЗДІЛ 5. ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було проведене комплексне дослідження реалізації наскрізного шифрування на основі протоколу Signal в одному з найпопулярніших месенджерів «WhatsApp».

Робота включала як теоретичний аналіз архітектури протоколу, так і практичну частину, що полягала у зворотній розробці мобільного iOS додатку WhatsApp.

У межах дослідження було опрацьовано теоретичну базу, а саме вивчено структуру Signal протоколу, його архітектуру, принципи побудови асинхронних безпечних сесій, механізми прямої секретності з використанням подвійного рачету та обміну ключами.

Було проведено практичний статичний та динамічний аналіз застосунку WhatsApp. Було вивчено структуру виконуваного файлу, структуру класів, проведено ідентифікацію ключових компонентів, використовуючи IDA. Було проведено трасування функцій та детальне динамічне дослідження окремих важливих функцій, шляхом дослідження їх стеку викликів, параметрів, тощо.

В ході дослідження були визначені точки входу до важливих криптографічних функцій, що дало повну комплексну картину внутрішньої реалізації протоколу Signal у WhatsApp. У результаті було підтверджено відповідність реалізації протоколу його специфікації у випадку індивідуальних чатів.

Також було випрацювано методологію, яку можна адаптовувати для аналізу шифрування у інших месенджерах з закритим вихідним кодом.

# ВИКОРИСТАНІ ДЖЕРЕЛА


1. Jonathan Levin “*OS Internals (Mac OS X & iOS Internals, 2nd Edition) Volume I: User Space”. 2019
2. Jonathan Zdziarski “Hacking and Securing iOS Applications” : O’Reilly Media Inc. 2012
3. https://signal.org/docs/specifications/x3dh/ - офіційна специфікація адаптації X3DH до протоколу Signal
4. https://signal.org/docs/specifications/doubleratchet/ - офіційна специфікація адаптації Double Ratchet до протоколу Signal
5. https://github.com/signalapp/libsignal-protocol-c/tree/master - офіційний репозиторій реалізації протоколу Signal на мові C