Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

РЕАЛІЗАЦІЯ АЛГОРИТМІВ ПОСТКВАНТОВОЇ КРИПТОГРАФІЇ ПРИ ПОТОКОВОМУ ШИФРУВАННІ ДАНИХ У СИСТЕМАХ «КЛІЄНТ-СЕРВЕР»

Виконав:
здобувач IV курсу
групи ІПЗ-41
спеціальності 121 «Інженерія
програмного забезпечення»
Тишков Олександр Ігорович

Науковий керівник:
кандидат юридичних наук, доцент,
Кіндрат Павло Вадимович

Рівне – 2025

ЗМІСТ

ВСТУП

Сучасне суспільство все більше покладається на інформаційні процеси, які стають рушійною силою економіки, соціальних відносин і військової справи. Інформаційні процеси охоплюють процеси збору, підготовки, передачі, обробки, перетворення та використання інформації в різних сферах життя. [1] В таких умовах використання надійного захисту інформації що передається набуває особливого значення.

Захист даних у цифрових інформаційних системах потребує уваги з боку профільних спеціалістів яка постійно зростає. Водночас, через розвиток квантових технологій і комп'ютерів в сучасному світі набуває все більшої важливості постквантове шифрування. Квантові комп'ютери здатні значно прискорити компрометацію багатьох протоколів, які нині базуються на складних математичних задачах, таких як RSA, SHA-2 або Діффі-Хелмана.

Постквантові криптографічні протоколи спрямовані на забезпечення безпеки даних у умовах зростаючої загрози з боку квантових комп'ютерів. Ці протоколи ґрунтуються на математичних задачах, які квантові комп'ютери не можуть ефективно розв'язати, що робить їх більш надійними в порівнянні з традиційними методами криптографії. Важливо зазначити, що розробка постквантових криптографічних протоколів все ще перебуває на початковій стадії, і наразі не існує загальноприйнятих стандартів для таких рішень. Тому кожен новий протокол потребує ретельної перевірки та оцінки перед його широким впровадженням. Проте значення постквантової криптографії буде лише зростати, оскільки квантові комп'ютери стають потужнішими та більш доступними.

Метою цієї роботи є аналіз існуючих мережевих протоколів, що підтримують шифрування, а також їх порівняння. Крім того, буде проведено огляд та аналіз постквантових криптографічних систем. У рамках дослідження розробимо метрики для їх порівняння та оберемо конкретні протоколи для впровадження у власний мережевий протокол. Результатом стане бібліотека з

гнучким та універсальним API, яку можна буде легко інтегрувати в існуючі системи без значних змін.

Отже, ця робота має значне значення для розвитку криптографії та захисту даних від атак, що здійснюються за допомогою квантових комп'ютерів. Дослідження допоможе вирішити проблему безпеки даних у контексті загрози з боку квантових технологій, що є надзвичайно актуальним у наш час. Розробка власного мережевого протоколу, що використовує постквантові алгоритми, стане відповіддю на потребу в надійному захисті даних від атак як традиційних, так і квантових комп'ютерів. Новий протокол повинен забезпечити створення квантово-безпечного каналу зв'язку та бути здатним до інтеграції в існуючі системи, а також використовуватися при розробці нових. Для цього він має бути крос-платформним і гарантувати прийнятний рівень швидкості, оскільки однією з основних проблем постквантової криптографії є її швидкодія та ресурсомісткість. Таким чином, розробка ефективного квантостійкого мережевого протоколу є важливим завданням для забезпечення безпеки зв'язку в майбутньому.

Об'єктом дослідження є системи криптографічного захисту.

Предметом дослідження є методи забезпечення конфіденційності даних у системах «клієнт-сервер».

Мета дослідження полягає в підвищенні рівня захищеності мережевого зв'язку та забезпеченні захисту даних від атак, здійснюваних як звичайними, так і квантовими комп'ютерами, під час передачі даних через мережу з використанням постквантових криптографічних примітивів.

Методи дослідження:

– аналіз сучасних технологій шифрування даних: Цей метод передбачає детальний аналіз існуючих технологій та методів пост квантової криптографії в системах передачі даних. Він включає в себе вивчення принципів роботи, сильних і слабких сторін, а також можливостей подальшого вдосконалення.

– експериментальне тестування та оцінка реалізованого методу шифрування: Цей метод включає проведення експериментів для перевірки

ефективності та надійності розроблених методів шифрування даних. Він передбачає проведення тестів у реальних умовах або за допомогою симуляційних середовищ для оцінки їх працездатності та визначення можливих обмежень чи недоліків.

Практичне значення дослідження полягає в можливості впровадження розроблених алгоритмів пост квантового шифрування у реальні системи контролю доступу. Вдосконалення засобів криптографічного захисту може сприяти підвищенню рівня безпеки інформаційних систем, запобіганню несанкціонованого доступу до даних.

Крім того, результати дослідження можуть бути використані в інших галузях, де необхідний високий рівень мережевої безпеки.

Структура роботи. Дипломна робота складається зі вступу, трьох розділів, висновків, переліку використаних джерел та додатків. Загальний обсяг роботи становить 68 сторінок. Вона містить 8 рисунків та 12 таблиць. Список використаних джерел включає 39 найменувань. Обсяг додатків – 9 сторінок**.**

РОЗДІЛ 1.
ПОРІВНЯЛЬНИЙ АНАЛІЗ ПРОТОКОЛІВ МЕРЕЖІ

1.1.Огляд наявних протоколів

Мережеві протоколи з підтримкою шифрування забезпечують захист передачі даних між клієнтами в мережі від несанкціонованого доступу, перехоплення та підслуховування. Зазвичай такі протоколи включають кілька основних компонентів:

- протокол обміну ключами;
- протокол автентифікації сторін;
- протокол шифрування повідомлень;
- протокол перевірки цілісності.

Протокол обміну ключами забезпечує взаємний обмін ключами між сторонами, які здійснюють зв’язок. Ці ключі, або їх похідні, використовуються для шифрування та розшифрування повідомлень. Зазвичай для цього застосовується асиметричне шифрування, що дозволяє кожній стороні зберігати свій власний ключ і використовувати його для захисту даних під час передачі. Протокол автентифікації сторін гарантує, що учасники зв’язку в мережі є тими, за кого себе видають. Він може базуватися на різних принципах, таких як введення пароля, використання цифрових підписів або взаємодія з довіреним центром автентифікації.

Протокол шифрування повідомлень забезпечує захист передачі даних між двома сторонами шляхом їх шифрування. Зазвичай для цього застосовуються симетричні методи, такі як AES (Advanced Encryption Standard), які дозволяють шифрувати та розшифровувати інформацію за допомогою спільного ключа. Протокол перевірки цілісності гарантує, що повідомлення не зазнали змін під час передачі. Для цього використовуються хеш-функції, які створюють унікальний дайджест (короткий текст, що є високоімовірним представником первинного тексту будь-якої довжини) для кожного повідомлення, що дозволяє перевірити його цілісність на стороні отримувача.

Ці компоненти забезпечують безпечну передачу даних у мережі, захищаючи їх від несанкціонованого доступу, перехоплення та підслуховування. Тому протоколи з підтримкою шифрування є надзвичайно важливими для гарантування безпеки даних у мережі. Крім того, важливо розглянути основні аспекти безпеки інформації: конфіденційність, автентифікацію та цілісність даних. Ці три елементи разом забезпечують комплексний захист даних, запобігаючи несанкціонованому доступу інформації:

- конфіденційність – гарантує, що доступ до захищеної інформації мають лише авторизовані користувачі. Захист даних здійснюється за допомогою криптографії, яка шифрує інформацію під час передачі та розшифровує її при отриманні.

- автентифікація – передбачає перевірку ідентичності користувачів перед наданням їм доступу до захищеної інформації. Це може бути реалізовано через перевірку пароля, біометричні дані або інші методи ідентифікації.

- цілісність даних означає, що інформація повинна бути захищена від змін або пошкоджень під час передачі. Це можна забезпечити за допомогою механізмів контролю цілісності, таких як хешування або цифровий підпис.

Протоколи захищеного інформаційного обміну забезпечують надійну роботу інформаційних систем. Під інформаційними системами розуміємо будь-які системи, які за допомогою технічних засобів виконують одну або кілька з таких функцій: збирання, передавання, перетворення, накопичення, зберігання та обробка інформації. [2]

Один із найпоширеніших протоколів, що підтримує шифрування, – це SSL/TLS (Secure Sockets Layer/Transport Layer Security). Він призначений для захисту даних в Інтернеті, зокрема під час виконання онлайн-операцій, таких як платіжні транзакції та передача конфіденційної інформації. SSL/TLS забезпечує шифрування даних і автентифікацію сторін, що встановлюють з’єднання.

Іншим протоколом, що підтримує шифрування, є IPsec (Internet Protocol Security). Він призначений для захисту мережевого трафіку, який передається

між комп’ютерами в Інтернеті. IPsec забезпечує шифрування, аутентифікацію та контроль цілісності даних, що передаються через мережу.

Для захисту передачі даних у бездротових мережах часто використовують протоколи з підтримкою шифрування, такі як WPA (Wi-Fi Protected Access) та WPA2. Ці протоколи забезпечують шифрування трафіку та автентифікацію користувачів, які підключаються до бездротової мережі.

Крім того, існують різні протоколи, які підтримують шифрування для захисту певних типів мережевих з’єднань. Наприклад, SSH (Secure Shell) забезпечує безпечний доступ до віддалених комп’ютерів, а PGP (Pretty Good Privacy) використовується для захисту електронної пошти.

Давайте детальніше розглянемо протоколи SSL/TLS та IPsec, які є найпоширенішими та добре зарекомендували себе. Проведемо порівняльний аналіз, щоб визначити, в якому напрямку слід рухатися при розробці власного протоколу, а також виявимо його сильні та слабкі сторони. Зосередимося на ключових аспектах, на які варто звернути увагу під час розробки. Для порівняння протоколів можна використовувати різні критерії, і в результаті огляду було обрано наступні.

- незалежність протоколу від апаратного забезпечення – чи впливає швидкість роботи протоколу на апаратну частину;
- рівень криптографічної стійкості – чи гарантує протокол високий рівень захисту;
- автентифікація сторін – чи передбачає протокол автентифікацію учасників;
- швидкість обміну даними – наскільки швидко відбувається процес handshake;
- рівень мережі – на якому рівні мережевої архітектури функціонує протокол;
- складність налаштування – наскільки важко налаштувати системи;
- підтримка UDP – чи підтримує протокол User Datagram Protocol (UDP);

- необхідність додаткового програмного забезпечення – чи потрібно додаткове ПЗ для роботи з протоколом;

- перевірка цілісності – чи здійснюється перевірка цілісності повідомлень;

- поширеність: чи є протокол загальнодоступним, чи він призначений для специфічних випадків.

- стиснення даних: чи підтримується стиснення для зменшення витрат на трафік.

- набір підтримуваних шифрів: чи має протокол широкий вибір шифрів.

1.2.Протокол SSL/TLS

Протокол TLS (Transport Layer Security) [3] – це криптографічний протокол, який забезпечує безпечну передачу даних через Інтернет. TLS є наступником протоколу SSL (Secure Sockets Layer) [3] і використовується для захисту різних типів з’єднань, включаючи веб-сайти, електронну пошту, миттєві повідомлення та інші.

TLS функціонує на рівні транспортного протоколу, наприклад, TCP, і забезпечує конфіденційність, цілісність та автентичність даних за допомогою шифрування та ідентифікації сторін, що спілкуються. Під час встановлення з’єднання TLS клієнт і сервер взаємодіють для узгодження параметрів шифрування, таких як тип шифрування та ключі, які будуть використовуватися для захисту даних під час їх передачі. Процес TLS складається з кількох етапів, включаючи:

- Handshake – це процес взаємодії між клієнтом і сервером, що дозволяє узгодити параметри шифрування та взаємну ідентифікацію сторін.

- Record Protocol – протокол, який гарантує шифрування та цілісність даних, що передаються через з’єднання.

- Alert Protocol – протокол, призначений для обробки помилок та повідомлень про них.

TLS представлений кількома версіями: TLS 1.0, TLS 1.1, TLS 1.2 та TLS 1.3. Кожна з цих версій має свої параметри шифрування та інші особливості. Найсучасніша версія, TLS 1.3, забезпечує швидший і безпечніший обмін даними, зменшуючи кількість з’єднань під час розгортання та покращуючи підтримку шифрування. У моделі OSI TLS не відноситься до конкретного шару, але має характеристики, що охоплюють як транспортний (4-й рівень), так і презентаційний (6-й рівень) шари.

Перед початком обміну даними через TLS клієнт і сервер повинні узгодити параметри з’єднання. Це включає вибір версії протоколу, метод шифрування даних, а також перевірку сертифікатів за необхідності. Процес ініціації з’єднання називається TLS Handshake.

Процес встановлення з’єднання є критично важливим і вразливим етапом при використанні протоколу TLS, оскільки саме в цей момент застосовуються алгоритми асиметричного шифрування даних. Розглянемо детальніше кожен крок цієї процедури (узагальнену схему можна побачити на рисунку 1.1).

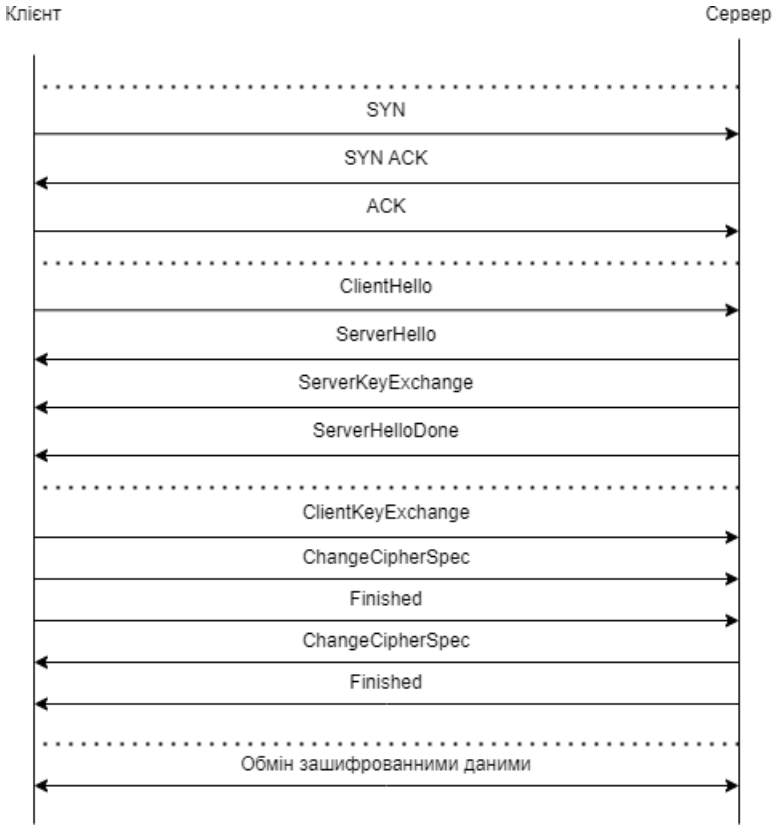

Рисунок 1.1 – Загальний принцип роботи протоколу TLS 1.2

Крок 1. Оскільки TLS функціонує на основі TCP, спочатку між клієнтом і сервером встановлюється TCP-з’єднання [3]. Процес виглядає наступним чином: – клієнт надсилає повідомлення про намір встановити з’єднання – SYN; – сервер відповідає, підтверджуючи можливість встановлення з’єднання – SYN ACK; – клієнт підтверджує отриману відповідь від сервера, після чого обидві сторони створюють стабільне з’єднання, що дозволяє розпочати фактичний процес передачі даних – ACK.

Крок 2. Клієнт підтверджує своє бажання встановити з’єднання з сервером і надсилає йому повідомлення ClientHello. У цьому повідомленні клієнт також включає список запропонованих наборів шифрів та бажану версію TLS для використання. Крім того, клієнт генерує випадкове число, яке буде використовуватися пізніше (воно відоме як «client random»), і перевіряє, чи є це сеансом відновлення. Якщо це не так, переходить до третього кроку.

Якщо клієнт надсилає повідомлення ClientHello з позначкою про відновлення сеансу, сервер може скористатися попередніми ключами, які використовувалися в минулому сеансі, для встановлення нового з’єднання. У відповідь на отримане повідомлення ClientHello з позначкою про відновлення, сервер надсилає повідомлення ServerHello, яке містить випадкове число сервера (так зване «server random») та підтвердження використання попередніх ключів для відновлення з’єднання.

Після цього сервер і клієнт здійснюють додатковий обмін інформацією та підтвердженнями для перевірки легітимності та генерації нових ключів, які забезпечать захист даних, що передаються в новому сеансі.

Крок 3. Сервер підтверджує прийнятні параметри для підключення та генерує випадкове число, яке буде використано пізніше (це число називається «server random») – ServerHello.

Крок 4. Сервер надсилає клієнту сертифікат відкритого ключа для автентифікації, відомий як ServerKeyExchange. Під час встановлення TLS-з’єднання клієнт перевіряє, чи сертифікат підписаний довіреним центром сертифікації (ЦС). Це необхідно для підтвердження, що сервер, з яким він

взаємодіє, дійсно є тим, за кого себе видає, а також для забезпечення захисту переданої інформації від несанкціонованого доступу. Якщо сертифікат є самопідписаним або недійсним, це може свідчити про можливу небезпеку з'єднання, і в такому випадку клієнт може відмовитися від з'єднання або показати попередження про потенційний ризик.

Крок 5. Сервер надсилає повідомлення про завершення роботи – ServerHelloDone.

Крок 6. Клієнт генерує попередній головний ключ (так званий «pre-master key») і надсилає його на сервер. Pre-master зашифровано відкритим ключем сервера, який отримується з сертифіката, раніше наданого сервером. Оскільки використовується асиметричне шифрування, лише сервер може розшифрувати це повідомлення. Клієнт обчислює master ключ – сесійний ключ, використовуючи попередньо згенеровані випадкові числа (client random + server random) та pre-master key.

Крок 7. Клієнт погоджується на подальше використання майстер-ключа та змінює метод шифрування на симетричний – ChangeCipherSpec.

Крок 8. Клієнт надсилає повідомлення про завершення роботи – Finished.

Крок 9. Після отримання pre-master ключа сервер використовує свій закритий ключ для його розшифрування. Цей процес гарантує, що обидві сторони є тими, за кого себе видають. Сервер обчислює master key, використовуючи попередньо згенеровані випадкові числа (client random + server random) разом із розшифрованим pre-master ключем. Відтепер сервер погоджується використовувати master key і переходить до симетричного методу шифрування, активуючи ChangeCipherSpec.

Крок 10. Остаточне повідомлення «Finished», яке надсилає сервер, є першим зашифрованим повідомленням, переданим за допомогою спільно обчисленого «master key». Це ж ключ використовується в алгоритмі коду автентифікації повідомлення (MAC) для підтвердження, що повідомлення не зазнало змін.

Варто зазначити, що шифрування з відкритим ключем застосовується лише під час процедури TLS Handshake. Це пов’язано з тим, що асиметричні системи мають продуктивність, яка в $(n * 1000)$ разів нижча, ніж у симетричних шифрів, хоча вони забезпечують безпечний обмін ключами між сторонами. Після встановлення захищеного з’єднання, яке забезпечується шифруванням і дешифруванням між двома клієнтами в мережі, переходять до симетричної криптографії. Спілкування в межах сесії шифрується за допомогою встановленого симетричного шифру, що необхідно для підвищення швидкості, оскільки криптографія з відкритим ключем вимагає значно більше обчислювальних ресурсів. Після завершення взаємодії з’єднання закривається.

Порівняння рукостискання між версіями TLS 1.3 та TLS 1.2 представлено на рисунку 1.2.

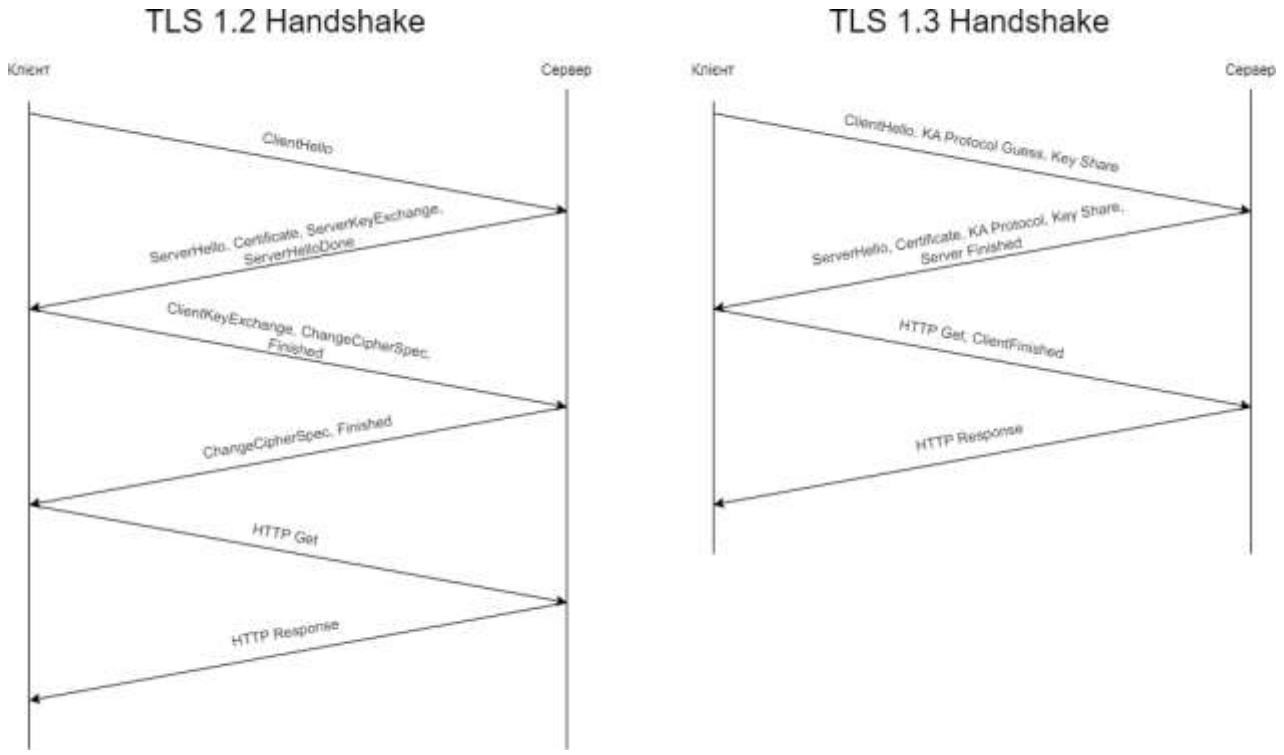

Рисунок 1.2 – Різниця «рукостискання» між TLS 1.2 та TLS 1.3

TLS 1.3 також пропонує ще швидшу версію рукостискання. Якщо клієнт і сервер вже раніше з’єднувалися (наприклад, користувач відвідував веб-сайт), і в них є спільний ключ, відновлення безпечного з’єднання вимагає на один обмін повідомленнями менше. Це стосується з’єднань, де клієнт і сервер мають

попередньо узгоджений спільний ключ (PSK), отриманий або ззовні, або під час попереднього рукостискання. Основна ідея полягає в тому, що після встановлення сеансу клієнт і сервер можуть отримати новий master key, використовуючи вже існуючий. Проте перший потік даних все ще шифрується за допомогою PSK, а не за допомогою нещодавно обчисленого свіжого спільного ключа, який не має прямої секретності у разі його компрометації.

Cipher Suite – це комплекс криптографічних алгоритмів, що забезпечують конфіденційність, цілісність та аутентифікацію даних у протоколі TLS. Під час рукостискання клієнт і сервер обмінюються пріоритетними списками наборів шифрів та обирають той, який найкраще підтримується обома сторонами.

У загальному, виділяють 4 основні набори шифрів: алгоритми обміну ключами, автентифікації (алгоритми цифрового підписа), алгоритми масового шифрування та алгоритми для автентифікації повідомлень. В таблиці 1.1 наведено основні представники кожної групи

Таблиця 1.1 Підтримувані набори шифрів

Обмін ключами	Автентифікації	Масового шифрування	Автентифікація повідомлень
RSA	RSA	AES	SHA-*
DH	ECDSA	CHACHA20	POLY1305
ECDH (ECDHE)	DSA	Camellia	MD5
DHE		ARIA	

Хоча TLS є ефективним і широко застосовуваним протоколом для захисту даних, він має свої вразливості та може піддаватися різним атакам. Ось кілька найбільш поширених слабкостей TLS [3]:

- Використання застарілих версій протоколу. Старі версії протоколу, такі як TLS 1.0 та 1.1, були офіційно визнані застарілими ще в 2018 році. Проте, їх все ще підтримують близько 35% користувачів. Хоча цей відсоток поступово зменшується, тенденція є позитивною і має продовжуватися [4]. Ці версії мають відомі проблеми з безпекою.

- Уразливості в криптографічних алгоритмах – деякі алгоритми шифрування та хешування, що застосовуються в TLS, можуть бути піддані різним атакам, зокрема атакам, що використовують колізії хеш-функцій.

- Слабкість паролів та інших засобів автентифікації – якщо клієнти та сервери використовують ненадійні паролі, це може дати змогу зловмисникам легко підібрати пароль і отримати доступ до захищених даних.

- Атаки на сертифікати – зловмисники можуть застосовувати підроблені сертифікати для перехоплення трафіку та здійснення MITM-атак [3].

Слід зазначити, що TLS не забезпечує абсолютної безпеки даних і не гарантує захисту від усіх можливих атак. Як і в будь-якій іншій системі захисту, ймовірність зламу TLS залежить від різних чинників, таких як навички та ресурси зловмисника, складність паролів і інших методів аутентифікації, що використовуються, та інші аспекти. Таким чином, TLS є ефективним протоколом для захисту даних, але для досягнення максимального рівня безпеки його потрібно правильно налаштувати та комбінувати з іншими засобами захисту. Важливо регулярно оновлювати сертифікати TLS і перевіряти їх дійсність, щоб уникнути можливих загроз безпеці. Додатково, можна використовувати віртуальні приватні мережі (VPN) для шифрування трафіку між різними системами. Загалом, для підтримки високого рівня захисту даних важливо використовувати комбінацію різних засобів і налаштовувати їх відповідно до власних потреб.

1.3. IPsec

IPsec – це набір протоколів і алгоритмів, розроблених для забезпечення безпеки мережевих комунікацій на рівні IP. Він захищає окремих користувачів, а також забезпечує автентифікацію доступу до даних і їх шифрування. На відміну від TLS, IPsec працює на третьому, мережевому рівні моделі OSI. IP-пакети, що передаються, захищені таким чином, що це залишається прозорим для мережевих додатків та інфраструктури.

IPsec може бути використаний для захисту різних протоколів, таких як TCP, UDP, ICMP та інших. Завдяки IPsec забезпечується безпека підключень до віддалених мереж, віртуальних приватних мереж (VPN), а також для передачі голосу та інших додатків, які потребують захисту даних під час їх пересилання через мережу. Протокол підтримує різні методи шифрування, включаючи AES, Blowfish, Triple DES, ChaCha та DES-CBC. IPsec використовує як асиметричне, так і симетричне шифрування, що дозволяє досягти швидкості та безпеки при передачі даних. У випадку асиметричного шифрування ключ шифрування стає публічним, тоді як ключ для дешифрування залишається приватним. Симетричне шифрування, в свою чергу, використовує один і той же закритий ключ для шифрування та розшифрування даних. IPsec спочатку встановлює безпечне з'єднання за допомогою асиметричного шифрування, а потім переходить на симетричне шифрування для підвищення швидкості передачі даних завдяки значно кращій продуктивності.

Архітектура IPsec [5] включає компоненти, які забезпечують три основні послуги безпеки: конфіденційність, автентифікацію та цілісність даних. Розглянемо основні елементи більш детально. Протокол ESP (Encapsulating Security Payload) [3] відповідає за забезпечення конфіденційності даних, що передаються через мережу. Він шифрує дані IP-пакета, захищаючи їх від несанкціонованого доступу. ESP також може виконувати функції автентифікації (AH). При його використанні можна вказати бажані функції, що дозволяє гнучко налаштовувати систему. Реалізація відбувається двома способами:

- ESP з опціональною автентифікацією;
- ESP у поєднанні з автентифікацією

Протокол AH (Authentication Header) [3] забезпечує аутентифікацію та цілісність даних, що передаються через мережу. Він додає до IP-пакета заголовок, який містить інформацію про аутентифікацію, цілісність (дозволяючи отримувачу перевірити, чи були дані пакета змінені під час передачі) та захист від повторного відтворення (запобігаючи несанкціонованій

передачі пакетів). Основними недоліками є відсутність шифрування та незахищеність конфіденційності даних.

Протокол IKE (Internet Key Exchange) [3] призначений для встановлення ключів шифрування та аутентифікації між двома пристроями, що взаємодіють. IKE реалізує це через серію обмінів ключами, створюючи безпечний і надійний тунель між клієнтом і сервером, що дозволяє їм легко та безпечно передавати зашифрований трафік. Безпека цього тунелю ґрунтується на методі обміну ключами Діффі-Хеллмана, який є одним із найпоширеніших методів забезпечення безпеки.

Протокол ISAKMP (Internet Security Association and Key Management Protocol) [3] є складовою частиною протоколу IKE. Його основне призначення полягає у встановленні ключів, автентифікації та узгодженні асоціацій безпеки для забезпечення безпечного обміну пакетами. Цей протокол визначає параметри безпеки, які регулюють, як дві системи можуть взаємодіяти одна з одною. Кожна асоціація безпеки встановлює з’єднання в одному напрямку, від одного користувача до іншого. Вона включає всі необхідні атрибути для з’єднання, такі як криптографічний алгоритм, режим IPsec, ключ шифрування та інші параметри, що стосуються передачі даних, необхідні для створення безпечного з’єднання.

SA (Security Association) [3] – це набір параметрів, що визначають з’єднання між двома пристроями. До цих параметрів належать використовувані протоколи, алгоритми, ключі, номери пакетів та інші характеристики. Кожен SA реалізує один режим і протокол; отже, якщо для обробки одного пакета потрібно використовувати два протоколи (наприклад, AH і ESP), знадобиться два SA. Варто також зазначити, що протокол IPsec функціонує в двох режимах, які забезпечують різний рівень захисту. Тунельний режим (tunnel mode) передбачає шифрування всього IP-пакета та додавання нового заголовка IP. Це дозволяє передавати зашифровані дані через відкриту мережу, підвищуючи захист інформації від несанкціонованого доступу. Режим тунелювання використовується для захисту цілих мереж або тунелів між двома мережами.

Транспортний режим (transport mode), який також відомий як наскрізне шифрування (end-to-end encryption), є методом захисту трафіку на рівні пакетів. У цьому режимі шифрується лише тіло IP-пакету, тоді як заголовок IP залишається відкритим. Незашифрований заголовок дозволяє маршрутизаторам визначати адресу призначення кожного пакета даних. Транспортний режим використовується для захисту трафіку між двома пристроями, які взаємодіють у надійній мережі, наприклад, для забезпечення безпеки прямого з'єднання між комп'ютерами. Однак, недоліком цього режиму є відсутність механізмів для приховування конкретних відправників і одержувачів пакетів, а також можливість проведення аналізу трафіку. Такий аналіз може надати інформацію про обсяги та напрямки передачі даних, інтереси абонентів і місцезнаходження керівників.

Протокол включає багато компонентних технологій і методів шифрування. Проте роботу IPSec можна розбити на п'ять основних етапів, які наведені на рисунку 1.3.

Рисунок 1.3 – Алгоритм роботи протоколу IPsec

Крок 1 – Визначення трафіку, що вимагає захисту. Процес полягає у визначенні трафіку, що потребує захисту за допомогою IPSec, тобто «цікавого трафіку», а також трафіку, який можна передавати відкрито. Системний адміністратор має встановити, який трафік може бути надісланий без захисту, а який вимагає додаткового захисту. Якщо ініціалізація IPSec пройшла успішно, переходять до фази 1 IKE.

Крок 2 – Фаза 1 IKE або раунд переговорів. Основна мета фази 1 IKE полягає в автентифікації однорангових вузлів IPsec та налаштуванні безпечного каналу між ними для забезпечення можливості обміну даними IKE. Фаза 1 IKE виконує такі функції:

- Однорангові вузли автентифікуються за допомогою сертифікатів або попередньо узгодженого секрету. – Визначає відповідну політику IKE SA між одноранговими вузлами для забезпечення захисту обміну IKE.
- Виконує автентифікований обмін Діффі-Хеллмана, в результаті якого отримуються відповідні спільні секретні ключі.
- Встановлює захищений тунель для узгодження параметрів «фази 2» IKE.

Оскільки основний режим забезпечує безпечний канал для передачі даних, він є значно безпечнішим, ніж агресивний режим. У агресивному режимі хост-ініціатор забороняє проведення переговорів і вимагає використання IKE SA, що призводить до меншої кількості обмінів і пакетів. Під час першого обміну більшість інформації вміщується в запропоновані значення IKE SA:

- відкритий ключ Діффі-Хеллмана;
- одноразовий криптографічний номер (nonce), підписаний контрагентом;
- ідентифікаційний пакет, що може бути використаний для перевірки особи через третю сторону.

Одержувач повертає всі необхідні елементи для завершення обміну. Залишається лише підтвердити обмін ініціатору. Недолік використання агресивного режиму полягає в тому, що обидві сторони обмінюються інформацією ще до встановлення безпечного каналу.

Крок 3 – IKE Фаза 2 або схема IPsec. На цьому етапі відбувається створення каналу IPsec через безпечний шлюз, який був налаштований під час IKE Фази 1. Кінцеві користувачі IPsec узгоджують алгоритми шифрування даних, які будуть використовуватися. Вони також визначають і розподіляють ключі для шифрування та дешифрування, необхідні для зв'язку в захищеній мережі. Крім того, користувачі передають криптографічні нонси – випадкові або псевдовипадкові числа, які використовуються лише один раз для генерації нового спільного секретного ключа, що запобігає повторним атакам і створенню фальшивих SA. Результатом цієї фази є IPsec SA.

Крок 4 – Зашифрований тунель IPsec. Після завершення другої фази та встановлення протоколів SA IPsec, інформація починає обмінюватися через тунель IPsec у швидкому режимі. Пакети шифруються та розшифровуються за допомогою шифрування, визначеного в IPsec SA.

Крок 5 – Завершення тунелю. Тунель IPsec закривається. Секретні асоціації (SA) можуть завершитися через тайм-аут, коли мине певна кількість секунд, або коли через тунель пройде визначена кількість байтів. Після завершення передачі даних хости видаляють закриті ключі, які використовувалися під час комунікації. Коли виникає потреба в нових SA, IKE проводить нову «фазу 2» і, за необхідності, нову «фазу 1» узгодження. Результатом успішних переговорів є нові SA та нові ключі. Нові SA можуть бути встановлені до закінчення терміну дії існуючих, що дозволяє безперервно підтримувати певний потік даних.

Під час переговорів визначаються два ключові параметри: алгоритм шифрування та алгоритм хешування. Алгоритм шифрування забезпечує безпосередній захист даних під час взаємодії між кінцевими користувачами, тоді як алгоритм хешування використовується для перевірки цілісності повідомлень.

Протокол підтримує широкий спектр алгоритмів, включаючи як застарілі та менш надійні (наприклад, DES, який використовується лише в ситуаціях,

коли потрібне менш надійне шифрування), так і нові криптостійкі алгоритми, що зарекомендували себе (такі як AES-GCM-* та SHA-*).

IPsec – це набір протоколів, призначених для забезпечення безпеки мережевого трафіку на рівні IP. Головною метою IPsec є захист мережі від атак, таких як перехоплення, підробка та модифікація переданих даних. Проте, цей алгоритм має й деякі вразливості, серед яких:

- конфігураційні складнощі – вимагає правильної налаштування, включаючи ключі, алгоритми шифрування та автентифікації, що може бути складним завданням для великих мереж. – Проблеми з сумісністю – IPsec може викликати труднощі в деяких мережевих середовищах, зокрема в мережах з численними точками входу/виходу або тих, що використовують Network Address Translation (NAT).

- витік інформації: неналежне зберігання або управління ключами може призвести до витоку конфіденційних даних, оскільки зловмисники можуть зламати ключі та отримати доступ до захищеної інформації.

- недостатній захист від DoS-атак: IPsec не забезпечує адекватного захисту від DoS-атак, які можуть перевантажити мережу та загрожувати її безпеці.

- масштабованість: масштабування на великих мережах може бути проблематичним, оскільки це призводить до значного збільшення обсягу даних, які потрібно зашифрувати та розшифрувати, що, в свою чергу, негативно впливає на швидкість роботи мережі.

- обмеження в управлінні доступом: протокол не забезпечує механізмів для детального контролю доступу до мережевих ресурсів, що робить його недостатнім для повного захисту мережі від несанкціонованого доступу. – Стійкість до криптоаналізу: хоча IPsec вважається досить стійким до криптоаналізу, все ж існують атаки, які можуть бути здійснені з використанням знань про структуру протоколу [6].

- вразливість до атак, що базуються на соціальній інженерії – протокол не здатен захистити від атак, які використовують соціальну інженерію, таких як

фішинг або видавання себе за іншу особу з метою отримання доступу до захищених даних.

Отже, IPsec є ефективним засобом забезпечення безпеки на рівні IP, проте має й деякі вразливості, які слід враховувати при його використанні. Для досягнення оптимальних результатів у забезпеченні безпеки можна застосовувати додаткові інструменти, такі як IKE-Scan або Nmap, для сканування конфігурацій IPsec та виявлення потенційних проблем безпеки.

1.4. Порівняння розглянутих протоколів

TLS та IPsec – це дві різні технології, призначені для захисту даних у мережах. Кожен з цих протоколів має свої переваги та недоліки, що робить їх придатними для різних сценаріїв використання. Обидва протоколи забезпечують надійне шифрування та аутентифікацію для захисту інформації. Проте IPsec має певні переваги, зокрема можливість захисту всієї мережі, а не лише окремих з'єднань. Крім того, IPsec може бути використаний для захисту будь-якого типу трафіку, тоді як TLS обмежений лише трафіком, що передається через протоколи, які підтримують TLS. Це обмеження пов'язане з розташуванням протоколів на різних рівнях моделі OSI, що зменшує можливості застосування TLS в інших сценаріях.

Врешті-решт, вибір між IPsec і TLS визначається конкретним сценарієм використання та вимогами до безпеки. Було проведено порівняльний аналіз характеристик і послуг інформаційної безпеки, використовуючи раніше обрані критерії. Результати цього аналізу представлені в таблиці 1.2.

Таблиця 1.2 Порівняння оглянутих протоколів

Критерій	TLS	IPsec
Незалежність протоколу від апаратної частини	+	-
Рівень криптографічної стійкості	Від помірного до сильного	Сильний
Автентифікація сторін	+	+

Час рукостискання	Швидкий	Повільний
Рівень мережі	Від транспортного до презентаційного	Мережевий
Складність налаштування	Легко	Складно
Підтримка UDP	DTLS	+
Необхідність додаткового ПЗ	-	+
Перевірка цілісності	+	+
Застосування	Наскрізна безпека між програмами	Весь трафік або наскрізне шифрування
Поширеність	+	-
Стистення даних	OpenSSL	+
Набір підтримуванних шифрів	Велика кількість, різних від версії до версії протоколу	Велика кількість, присутні слабкі алгоритми
Проблеми сумісності	-	NAT

Висновок до Розділу 1

У цьому розділі розглядаються популярні мережеві протоколи, що підтримують шифрування, зокрема TLS та IPsec. Обидва протоколи вважаються надійними для захисту даних, принаймні до тих пір, поки квантові комп’ютери не стануть поширеними. Квантові комп’ютери здатні розкрити ключі шифрування, які наразі використовуються в TLS та IPsec, що може дозволити зловмисникам отримати доступ до конфіденційної інформації, що передається через мережу. Це особливо стосується IPsec, оскільки він зазвичай використовує довгострокові ключі, які можуть бути скомпрометовані в епоху постквантових технологій.

IPsec зазвичай використовується для захисту даних на мережевому рівні, але також може бути застосований на рівні транспорту, забезпечуючи безпеку

всього трафіку, що проходить через нього. Це дозволяє захищати різноманітні протоколи та програми, проте вимагає більшої обчислювальної потужності для шифрування та дешифрування трафіку. Це може призвести до зниження продуктивності, особливо на пристроях з обмеженими ресурсами. Проте IPsec може використовувати апаратне прискорення для покращення продуктивності.
TLS є стандартом, що забезпечує захист комунікацій між клієнтом і сервером. Він оптимально підходить для наскрізної безпеки між програмами та широко використовується для захисту на транспортному рівні. Це означає, що TLS захищає лише трафік між двома кінцевими точками і не обробляє стільки ж трафіку, як IPsec. Крім того, TLS має нижчі криптографічні витрати в порівнянні з IPsec, що забезпечує кращу продуктивність на менш потужних пристроях. Проте TLS може бути вразливим до атак на прикладному рівні, таких як DDoS-атаки.

Отже, можна стверджувати, що використання TLS як основи для розробки протоколу має більше переваг. Хоча IPsec також є ефективним протоколом, його налаштування є більш складним і може вимагати додаткових апаратних ресурсів для досягнення оптимальної швидкості. TLS, будучи широко вживаним протоколом, має добре досліджений і гарантований рівень безпеки. Крім того, TLS забезпечує гнучкість у налаштуванні, що спрощує впровадження протоколу та інтеграцію з різними системами, а також забезпечує високу швидкість без потреби в додатковому апаратному прискоренні.

РОЗДІЛ 2.
ТЕХНОЛОГІЇ ПОСТКВАНТОВОЇ КРИПТОГРАФІЇ

2.1. Аналіз загроз для існуючих алгоритмів

Найбільшою загрозою для безпеки розглянутих мережевих протоколів є компрометація криптографічного алгоритму, що використовується під час їх роботи. Безпека криптографії ґрунтується на математичних «важких» проблемах – це функції, які легко обчислювати, але не мають зворотних обчислень. Практично всі алгоритми асиметричної криптографії базуються на задачах факторизації великих чисел (розкладання на множники) та дискретного логарифмування в різних алгебраїчних структурах.

Метою постквантової криптографії є створення криптографічних систем, які забезпечують захист від атак як квантових, так і класичних комп’ютерів, а також здатні інтегруватися з існуючими протоколами зв’язку та мережами. Ці алгоритми базуються на математичних задачах, які, на відміну від традиційних криптографічних методів, вважаються складними для розв’язання як класичними, так і квантовими комп’ютерами.

Квантові алгоритми здатні порушити безпеку існуючих алгоритмів, які базуються на обчислювальній складності певних математичних задач. Наприклад, популярна схема шифрування з відкритим ключем RSA залежить від складності розкладання великих цілих чисел на прості. Проте алгоритм Шора, квантовий алгоритм, може ефективно виконувати це розкладання, що потенційно загрожує безпеці шифрування RSA.

Алгоритм Гровера, ще один квантовий алгоритм, може бути застосований для прискорення пошуку рішень у несортованих базах даних. Це потенційно здатне зламати криптографічні хеш-функції, такі як SHA-256, які базуються на складності знаходження прообразу для заданого хешу.

Існують також алгоритми, що ґрунтуються на задачі дискретного логарифмування. Наприклад, алгоритм Ель-Гамаля може бути використаний для створення електронного підпису або шифрування даних. Його ефективність

базується на складності обчислення дискретного логарифму, яка вважається достатньо високою, щоб унеможливити розв’язання за прийнятний час. Однак у цій сфері також впливають квантові алгоритми, зокрема модифікація алгоритму Шора, адаптована для розв’язання задачі дискретного логарифмування.

Алгоритм Шора – це квантовий алгоритм, призначений для розкладання великих цілих чисел на прості множники. Факторизація є складною задачею для класичних комп’ютерів і становить основу криптостійкості багатьох криптографічних алгоритмів, таких як RSA.

Алгоритм Шора використовує квантову суперпозицію та інтерференцію для знаходження множників великих цілих чисел. Він починає з поміщення вхідного числа в стан суперпозиції, що дозволяє системі перебувати у всіх можливих станах одночасно, кожен з яких має свою ймовірність. Далі застосовується модульна функція піднесення до вхідного регістру, а також використовується квантова інтерференція – явище, яке дозволяє усувати неправильні відповіді, підкреслюючи правильні. Після цього класичні алгоритми аналізують періодичність вихідних значень і визначають множники вхідного числа.

Наприклад, розглянемо атаку, яка націлена на алгоритм RSA (модуль N вираховується як добуток двох великих простих чисел p та q). Суть цієї атаки полягає у пошуку ключа шляхом факторизації. Якщо ми зможемо обчислити прості числа, отримаємо доступ до числа Ейлера $\varphi(n)$, звідки – до обчислення приватного ключа за значенням публічного. Задача алгоритму – для непарного композитного числа N потрібно знайти ціле число d (від 1 до N), на яке ділиться N.

Крок 1: обираємо будь-яке випадкове число $r < N$ таке, що r і N – взаємно прості числа

Крок 2: квантовий комп’ютер використовується для визначення невідомого періоду p функції $f_{r,N}(x) = r^x mod N$.

Крок 3: якщо p – непарне ціле число, повертаємося до кроку 1. В іншому

випадку переходимо до наступного кроку.

Крок 4: оскільки p – парне ціле число, то $(r^{p/2} - 1)(r^{p/2} + 1) = 0 mod N$.

Крок 5: якщо значення $r^{p/2} + 1 = 0 mod N$, то повертаємось до кроку 1.

Крок 6: якщо значення $r^{p/2} + 1 \neq 0 mod N$, переходимо до наступного кроку.

Крок 7: обчислюємо $d = gcd(r^{p/2} - 1, N)$, де gcd – greatest common divisor, або функція пошуку найбільшого спільного дільника. В результаті отримано один з дільників числа N.

Алгоритм дозволяє виконати факторизацію числа N за час O $(log^3 N)$, використовуючи O $(log N)$ кубітів, є ефективним для великих значень N, оскільки складність алгоритму залежить логарифмічно від N. У порівнянні з класичними алгоритмами, які мають експоненціальну складність, це робить його значно швидшим та більш ефективним для великих значень N. При використанні квантового комп'ютера з кількома тисячами кубітів стає можливим зламування криптографічних систем з відкритим ключем (наприклад, RSA). Алгоритм Шора здатний зробити злом не просто за поліноміальний час, а за час, який можна порівняти з часом множення простих чисел. Однак алгоритм вимагає великої кількості кубітів і наразі виходить за межі можливостей сучасних квантових комп'ютерів

Алгоритм Гровера [8] – це квантовий алгоритм, який можна використовувати для пошуку в несортованій базі даних за час O($\sqrt{N}$), де N – розмір бази даних, може бути використаний для швидкого пошуку значення хеш-функції, що відповідає певному вхідному повідомленню. Це може бути корисно для зловмисників, які намагаються зламати криптографічні хеш-функції, що використовуються для забезпечення цілісності даних

Хеш-функції – це односторонні функції, які широко використовуються в криптографічних протоколах для забезпечення цілісності та автентичності даних. Криптографічна хеш-функція приймає вхідне повідомлення будь-якої довжини і генерує вихідні дані фіксованого розміру, відомі як хеші. Ці функції розроблені так, що обчислювально складно знайти два різні повідомлення, які б

давали однакове хеш-значення, або змінити хеш-функцію для відновлення вихідного повідомлення з хешу.

Атаки на хеш-функції зазвичай реалізуються за допомогою методу перебору, що може бути дуже трудомістким, особливо при використанні хеш-функцій з великим розміром дайджесту. Наприклад, для атаки на першу половину (128 біт) результату функції SHA-256 знадобилося понад 7 місяців та близько 40 МВт електроенергії, використовуючи два біткоїн-майнера та суперкомп'ютер «Blue Gene/Q» [9]. Якщо хеш-функція застосовується для зберігання паролів, зловмисники можуть намагатися зламати пароль, перебираючи всі можливі комбінації. У випадку, коли хеш-функція використовується для захисту даних від несанкціонованих змін, зловмисники можуть скористатися алгоритмом Гровера для знаходження значення хеш-функції, яке дозволить їм змінити пакети даних, не порушуючи цілісність хеш-функції.

Алгоритм Гровера може бути застосований для виявлення колізій у хеш-функціях, тобто для знаходження двох різних повідомлень, які мають однакове хеш-значення. Цей алгоритм дозволяє досліджувати простір можливих повідомлень, здатних створити певне хеш-значення, і знаходити колізії за час $O(\sqrt{N})$, тоді як класичні алгоритми потребують $O(N)$, де N – кількість можливих повідомлень.

Отже, алгоритм Гровера може бути використаний для подолання колізійної стійкості хеш-функцій. Однак цю вразливість можна зменшити, застосовуючи хеш-функції з більшими розмірами виходу, що розширює простір пошуку та ускладнює виявлення колізій за допомогою алгоритму Гровера в прийнятні терміни. Також варто розглянути постквантові криптографічні алгоритми та квантово стійкі схеми підпису. Це може становити потенційну загрозу для криптографічних хеш-функцій, таких як MD5, SHA-1, SHA-2 та SHA-3. Наприклад, 512-бітна хеш-функція, така як SHA-512, в найкращому теоретичному випадку вимагала б 2^256 операцій для зламу за допомогою

алгоритму Гровера, що наразі перевищує можливості як класичних, так і квантових комп’ютерів.

Хоча алгоритм Шора найчастіше асоціюється з розкладанням великих цілих чисел, його також можна застосовувати для розв’язання задачі дискретного логарифмування в певних групах, зокрема в дискретних логарифмах над скінченними полями. Це є ще однією складною проблемою в класичній криптографії.

Задача дискретного логарифмування – це задача знаходження цілого числа x, яке задовольняє рівняння $g^x = h$, де g і h – елементи скінченної групи.Ця проблема є складною для класичних комп’ютерів і становить основу криптостійкості таких криптографічних алгоритмів, як Діффі-Хеллман та ElGamal. Ці системи широко використовуються на практиці, а їхня безпека ґрунтується на складності розв’язання задачі дискретного логарифмування за допомогою класичних комп’ютерів.

Крок 1: нехай g – генератор скінченої циклічної групи G порядку q. Дано $y = g^k \in G$, необхідно знайти значення k.

Крок 2: розглянемо функцію двох змінних f: $(x_1, x_2) \mapsto g^{x_1} y^{x_2}$.

Крок 3: Підпрограма пошуку періоду знаходить пару (ω_1, ω_2), таку, що $f(x_1 + \omega_1, x_2 + \omega_2) = f(x_1, x_2)$.

Крок 4:це означає що $g^{\omega_1} y^{\omega_2} = 1_G \iff g^{\omega_1 + k\omega_2} = 1_G$, отже $\omega_1 + k\omega_2 \equiv 0$ або $k\omega_2 \equiv -\omega_1 (mod(q-1))$ ($q-1$ за малою теоремою Ферма).

Крок 5: існує q пар (ω_1, ω_2), які дають цей результат. Якщо кожен результат однаково вірогідний, то існує ймовірність $1/q$, що $(\omega_1, \omega_2) \equiv (0, 0)(mod(q-1))$. За ймовірності $(q-1)/q$, що вона не дорівнює нулю, розв’язок задачі дискретного логарифмування тоді визначається як

$k = -\omega_1/\omega_2 \cdot mod(q-1)$.

Отже, алгоритм Шора може бути застосований для розв’язання задачі дискретного логарифмування, використовуючи суперпозицію вхідних значень, модульну функцію піднесення до степеня та квантову інтерференцію для

виявлення періодичності вихідних значень. Після цього алгоритм переходить до класичних методів для обчислення дискретного логарифма, таких як алгоритми Шенкса, Поліга-Хелмана та GNFS. Хоча алгоритм Шора вимагає лише O($log^3 q$) і може бути використаний для зламу багатьох криптографічних систем, заснованих на задачі дискретного логарифмування, слід зазначити, що не всі групи є вразливими до атак за допомогою цього алгоритму, наприклад, мультиплікативна група залишків за модулем n.

Одним із можливих застосувань алгоритму Шора для розв'язання проблеми дискретного логарифмування є загроза безпеці протоколів TLS/SSL, які є основними засобами захисту Інтернет-зв'язку. TLS/SSL базується на узгодженні єдиного секретного ключа через обмін відкритими ключами між сторонами, використовуючи алгоритм Діффі-Хеллмана для встановлення спільного секрету. Безпека цього протоколу ґрунтується на складності розв'язання задачі дискретного логарифмування. Теоретично, успішне застосування алгоритму Шора дозволить зловмиснику перехоплювати зашифровані комунікації та потенційно викрадати конфіденційну інформацію. Однак, як і у випадку з факторизацією, наразі не існує достатньої обчислювальної потужності, щоб алгоритм працював ефективно в реальних умовах.

2.2. Постквантові криптографічні системи

Квантові обчислення мають потенціал для розв'язання складних математичних задач, на яких ґрунтується безліч алгоритмів. Тому для захисту від квантових атак важливо використовувати криптографічні системи, які є стійкими до таких загроз, тобто квантово-стійкі. Ці системи застосовують алгоритми, що базуються на математичних проблемах, для яких не існує ефективних методів криптоаналізу. Існує кілька типів квантово-стійких алгоритмів, які можуть бути використані для створення криптографічних систем, здатних витримувати атаки як з боку квантових, так і звичайних комп'ютерів:

- Lattice-based cryptography – криптографія на основі ґраток;
- Multivariate Cryptography – мультиваріативна криптографія;
- Hash-based cryptography – криптографія на основі хеш-функцій;
- Code-based cryptography – криптографія на основі кодів коригування помилок;
- Isogeny-based cryptography – криптографія на основі ізогенії;
- Symmetric key cryptography – симетрична криптографія (за умовою використання ключа великого розміру стійка до квантових алгоритмів)

Інформацію про алгоритми було отримано в результаті аналізу раундів відбору, дослідження та стандартизації квантовостійких алгоритмів, що проводиться Національним інститутом стандартів і технологій США (NIST) [11]. Одним із ключових проектів цієї організації є підпроєкт PQC (Post-Quantum Cryptography) [12]. У рамках проекту PQC NIST організовує конкурси для аналізу раундів відбору, дослідження та стандартизації квантовостійких алгоритмів.

NIST збирає пропозиції щодо постквантових криптосистем, запрошуючи фахівців представити свої алгоритми, а також коментарі від громадськості в рамках процесу оцінки. Організація планує провести кілька раундів відбору, щоб досягти найкращих результатів. Метою цього процесу є вибір ряду прийнятних кандидатів для стандартизації криптосистем. На даний момент NIST вже провела три раунди відбору, і наразі триває четвертий. Під час оцінки кандидатів проводиться ретельний аналіз поданих алгоритмів, який є відкритим і прозорим для громадськості. Криптографічна спільнота заохочується до проведення власних аналізів та оцінок, що сприяє неупередженості в оцінюванні та збільшує обсяги досліджень як щодо алгоритмів, так і в цій галузі загалом. NIST закликає рецензентів представити свої висновки та можливі практичні атаки як на версії з параметрами, що забезпечують повний рівень безпеки, так і на набори параметрів з нижчими рівнями безпеки. Тому для підкреслення значущості алгоритмів після їх розгляду буде наведено відомості про участь/перемогу в номінаціях від NIST.

NIST класифікує алгоритми за п’ятьма рівнями безпеки [12]. Ці рівні визначаються на основі теоретичного часу, необхідного для компрометації криптографічного протоколу за допомогою квантового комп’ютера. Для оцінки цього часу використовуються відповідні атаки: для симетричних блочних шифрів – атака, що порушує визначення безпеки, яка має вимагати обчислювальних ресурсів, порівнянних або більших, ніж ті, що потрібні для пошуку ключів; для хеш-функцій – атака, спрямована на пошук колізій. На сьогодні NIST визначає п’ять рівнів безпеки постквантових криптоалгоритмів наступним чином:

- алгоритм має бути настільки складно скомпрометувати, як і виконати пошук ключів у симетричному блочному шифрі з довжиною ключа у 128 біти (наприклад, AES128);
- має відповідати стійкості 256-бітної хеш-функції (наприклад, SHA256);
- має відповідати стійкості симетричного шифру з ключем довжиною 192 біти (наприклад, AES192);
- має відповідати стійкості 384-бітної хеш-функції (наприклад, SHA384);
- найвищий рівень, має забезпечувати захист еквівалентний до симетричного шифрування з ключем довжиною 256 бітів (наприклад, AES256).

1. Криптографія на основі ґраток – це вид криптографічної схеми, що ґрунтується на математичній теорії ґраток. Ґратки є дискретними математичними структурами, які зустрічаються в різних галузях математики, таких як теорія чисел, алгебра та геометрія. Вони представляють собою набір точок у n-вимірному просторі, розташованих за певним регулярним шаблоном, утворюючи сітку, що нескінченно простягається в усіх напрямках. Цей підхід може бути застосований у різних сферах, зокрема для обміну ключами, цифрових підписів та повністю гомоморфного шифрування.

Ґраткова криптографія заснована на обчислювальній складності ґратчастих задач, основою якої є задача найкоротшого вектора (SVP), яка вимагає знайти найкоротший ненульовий вектор у даній ґратці. Надано вхідну ґратку, представлену довільним базисом, і мета зловмисника полягає в тому,

щоб знайти найкоротший вектор від початку координат, коли задано основу ґратки (нульовий вектор не працює як відповідь) [12].

Найвідоміші атаки на криптографію, що базується на ґратках, спираються на класичні алгоритми, які виявляються неефективними та потребують експоненційно великих ресурсів. Найпопулярніший з цих алгоритмів має часову складність $O(2n\ logn)$ для точного розв'язання задачі про найкоротший вектор (SVP). Іншою важливою проблемою є задача найближчого вектора (CVP), яка полягає у знаходженні точки ґратки, найближчої до заданого цільового вектора. Найкращий відомий алгоритм для її розв'язання має складність $O(23.5n)$ [13, 14]. Криптографія на основі ґраток є відносно ефективною у реалізації та має потужні докази безпеки, що ґрунтуються на складності у найгіршому випадку. Незважаючи на значні дослідницькі зусилля, досі не відомо про ефективні квантові алгоритми, які могли б вирішувати основні задачі цього алгоритму значно швидше, ніж класичні. Єдина перевага квантових комп'ютерів у цьому контексті полягає в скромному загальному прискоренні.

Криптосистеми, що ґрунтуються на ґратках, зазвичай є алгоритмічно простими, ефективними та добре підходять для паралельної обробки. Алгоритми, які застосовуються в криптографії на основі ґраток, спираються на елементарні математичні операції, такі як множення матриць і модульна арифметика, що дозволяє їх легко реалізувати та оптимізувати на апаратному рівні.

2. Мультиваріативна криптографія (англ. multivariate cryptography, MC) – це вид криптографії з відкритим ключем, що ґрунтується на складності розв'язання систем багатовимірних поліноміальних рівнянь у кінцевих полях. У деяких випадках ці поліноми можуть бути визначені як у основному, так і в розширеному полі. Якщо поліноми мають ступінь два, їх називають багатовимірними квадратичними. Дослідження показали, що розв'язання систем багатовимірних поліноміальних рівнянь є NP-трудною задачею. Проте безпека MC вважається вразливою до алгебраїчних і диференціальних атак, а

також атак на базис Грьобнера. Як наслідок, MC не отримала широкого практичного застосування, і її використання залишається обмеженим.

У MC відкритий ключ представляє собою систему рівнянь багатовимірного полінома, тоді як закритий ключ – це знання коефіцієнтів цих поліномів. Для шифрування повідомлення відправник перетворює відкритий текст на систему поліноміальних рівнянь і розв'язує їх, використовуючи відкритий ключ. Одержувач, у свою чергу, застосовує свої знання закритого ключа для розв'язання тієї ж системи рівнянь, щоб розшифрувати повідомлення.

3. Криптографія, заснована на хешуванні, є формою постквантової криптографії, що використовує математичні властивості хеш-функцій для забезпечення безпеки. Основна концепція полягає в застосуванні односторонньої хеш-функції для генерації дайджесту повідомлення або хеш-значення з вихідного тексту. Це хеш-значення потім слугує цифровим підписом або ключем для шифрування та дешифрування інформації.

Головною перевагою хешування в криптографії є те, що це добре вивчена та широко застосовувана технологія, яка забезпечує високу стійкість до квантових атак. Це робить її перспективним варіантом для забезпечення довгострокової безпеки в епоху після квантових обчислень, за умови використання достатньо довгого ключа або дайджеста.

Проте криптографія, що базується на хешуванні, має певні потенційні недоліки. По-перше, безпека хеш-функції може бути скомпрометована, якщо зловмисники виявлять колізії, тобто два різних повідомлення, які генерують однакове хеш-значення. По-друге, для забезпечення безпеки криптографії на основі хешування потрібен відносно великий розмір ключа (який є результатом обчислення дайджесту повідомлення). Це пов'язано з тим, що алгоритм Гровера може бути використаний для зламу хеш-функції за час $O(\sqrt{n})$ (де n – довжина ключа), тоді як звичайні алгоритми мають експоненційну складність, $O(22)$, хоча вони ефективні лише для коротких ключів. Це призводить до уповільнення процесу генерації ключа або підпису, а також до збільшення

вимог до пам’яті та розміру фінального повідомлення для транспортування.

Криптографія, що базується на хешуванні, широко застосовується для створення цифрових підписів. Підпис на основі хешу (HBS) являє собою набір різних схем одноразового підпису. HBS використовує структуру даних у вигляді дерева для ефективного об’єднання кількох одноразових підписів. Для підписання повідомлення HBS обирає один з одноразових підписів зі своєї колекції та використовує його для підпису. Важливо, щоб алгоритм не застосовував один і той самий одноразовий підпис з колекції більше одного разу, оскільки це може загрожувати безпеці [21].

4. Криптографія, що ґрунтується на кодах коригування помилок (англ. code-based cryptography), базується на складності задачі декодування лінійного коду з виправленням помилок. Цей код може бути обраний з певною структурою або належати до конкретного сімейства, наприклад, квазіциклічних кодів або кодів Гоппа. Безпека цієї криптографії залежить від складності проблеми декодування, яка вимагає виявлення вектора помилки, що виник під час передачі кодового слова. Вважається, що ця задача є обчислювально складною (складність розв’язання задачі декодування кодів з виправленням помилок залишається NP-трудомісткою, хоча точна складність є предметом активних досліджень), що робить криптографію на основі кодів перспективним кандидатом для постквантової криптографії.

Класичним прикладом цієї системи є криптосистема McEliece, що базується на бінарному коді Гоппі. Classic McEliece [23] – це постквантова криптосистема з відкритим ключем, яка претендує на стандартизацію NIST і була запропонована Даніелем Дж. Бернштейном у 2017 році. Згідно з концепцією МакЕліса, відкритий ключ формується шляхом поєднання коду Гоппі та лінійного перетворення. Для шифрування повідомлення відправник додає до нього певну кількість випадкового «шуму», який можна усунути лише за допомогою коду Гоппі [24]. Відновлення повідомлення без знання відкритого ключа є складною обчислювальною задачею для зловмисника.

Одна з можливих проблем криптографії, що базується на кодах, полягає в тому, що відкритий ключ може бути підданий алгебраїчним атакам. У таких випадках зловмисник може отримати інформацію про секретний ключ, аналізуючи властивості відкритого ключа. Іншою проблемою є вразливість криптографії на основі кодів до атак з боку, коли зловмисник може спостерігати за процесом виконання алгоритму шифрування або дешифрування та витягувати інформацію про секретний ключ. Варто зазначити, що криптографія на основі кодів вимагає більших розмірів ключів. Це може призвести до подовження часу шифрування та дешифрування, а також потребувати більше місця для зберігання. Крім того, така криптографія може бути витратною з точки зору обчислень, оскільки процеси кодування та декодування вимагають значної кількості множень матриць, що може бути дорогим у плані обчислювальних ресурсів. Розглянемо квантово-безпечні алгоритми на основі кодів коригування помилок. Алгоритм Classic McEliece – це KEM, який відповідає всім п’яти рівням безпеки NIST. Час обчислення цього алгоритму є відносно швидким, проте він вимагає використання дуже великого розміру відкритого ключа (від 0.25 до 1.35 мегабайт), що є значним для передачі через канали зв’язку.

5. Ізогенія – це математичне відображення між еліптичними кривими, яке зберігає групову структуру точок на цих кривих. Криптографія, що ґрунтується на ізогенії (англ. isogeny-based cryptography), є відносно новою системою, з найвідомішим представником – протоколом обміну ключами Supersingular Isogeny Diffie–Hellman (SIDH). Цей підхід використовує відображення між еліптичними кривими для створення криптосистем з відкритим ключем. Безпека таких систем базується на одній з небагатьох складних математичних задач, яка наразі витримує атаки квантових комп’ютерів – на проблемах суперсингулярних ізогеній, що полягають у знаходженні ізогенії між двома суперсингулярними еліптичними кривими з однаковою кількістю точок.

Протоколи, що ґрунтуються на ізогенії, потребують значно менших ключів у порівнянні з іншими кандидатами пост-квантової криптографії. Проте, ці ключі все ще значно більші, ніж ті, що використовуються в алгоритмах на звичайних еліптичних кривих. Водночас продуктивність та можливості для більш складних криптографічних примітивів обмежені в порівнянні, наприклад, з системами, що базуються на ґратках [27].

Перший запропонований алгоритм, що базується на ізогеніях, використовував звичайні еліптичні криві, які ґрунтуються на складності задачі дискретного логарифмування еліптичної кривої. Проте, після аналізу можливих атак, з'ясувалося, що існують субекспоненціальні атаки на цей алгоритм, які можуть бути реалізовані за допомогою квантових комп'ютерів (атаку виявили А. Чайлдс, Д. Джао та В. Сухарєв [28]).

Однією з можливих проблем криптографії на основі ізогенії є те, що ця технологія є відносно новою та ще не повністю перевіреною. Це може призвести до наявності невиявлених вразливостей або слабких місць, які можуть бути використані зловмисниками. Крім того, хоча ключі для криптографії на основі ізогенії мають невеликі розміри, процес їх генерації все ще потребує значних обчислювальних ресурсів, що може стати недоліком у середовищах з обмеженими можливостями.

2.3. Порівняльне дослідження квантовостійких криптографічних алгоритмів

Щоб скомпрометувати сучасні криптографічні алгоритми, потрібен квантовий комп'ютер з мільйонами «кубітів». Наразі невідомо, коли такі комп'ютери стануть доступними. Проте стрімкий розвиток квантових комп'ютерних технологій вказує на те, що це може статися дуже скоро, і ми не можемо ігнорувати зростаючу загрозу з їхнього боку. Наприклад, у 2019 році компанія IBM створила квантовий процесор з 27 кубітами, а у 2021 році досягла вже 127 кубітів, що є вражаючим прогресом за всього два роки.

При розробці власного протоколу важливо обирати постквантові алгоритми, які забезпечують захист від можливих атак, що використовують квантові обчислення. Для цього потрібно вибрати конкретні криптографічні алгоритми, які будуть застосовуватися під час створення мережевого протоколу. Загалом, необхідно три криптографічні примітиви, на основі яких буде здійснено розподіл та порівняння розглянутих квантовостійких алгоритмів:

- алгоритм для отримання спільного секретного ключа – забезпечує можливість узгодження єдиного секретного ключа, який використовуватиметься для симетричного шифрування та дешифрування даних через відкритий канал зв’язку.

- алгоритм автентифікації сторін – дозволяє підтвердити, що сторони є тими, за кого себе видають, і що взаємодія з ними є передбачуваною.

- симетричний алгоритм захисту даних – застосовуватиметься на основному етапі спілкування (після узгодження спільного секретного ключа сеансу та додаткових перевірок за потреби) для захисту даних, оскільки він забезпечує значно вищу продуктивність у порівнянні з асиметричними алгоритмами.

В результаті аналізу існуючих мережевих протоколів у першому розділі було ухвалено рішення розробити новий протокол для постквантової криптографії, спираючись на принципи TLS, оскільки його можна буде легко розширити. Тому, аналізуючи примітиви, можна скласти пріоритетний список або виділити кандидатів з кожної категорії для подальшого використання, а також зрозуміти, чому їх можливо буде додати в майбутньому. На даний момент достатньо вибрати по одному алгоритму для кожного примітиву.

Аналіз продуктивності квантово-стійких алгоритмів здійснено в рамках проекту Open Quantum Safe (OQS) [34], який займається розробкою та прототипуванням таких алгоритмів. Бібліотека liboqs [35], створена OQS, містить відкритий код для квантово-стійких криптографічних алгоритмів. Основною метою OQS є стандартизація постквантової криптографії NIST для

механізмів шифрування ключів (KEM) та схем підпису. Крім того, OQS надає дані для порівняльного аналізу різних квантово-стійких алгоритмів, зокрема щодо їх продуктивності та споживання пам’яті [36]. Використання офіційних даних про швидкодію дозволяє уникнути похибок, тоді як інші критерії будуть сформовані відповідно до специфіки алгоритмів.

Алгоритм обміну ключами дає змогу двом сторонам, які ніколи не зустрічалися, узгодити спільний ключ. Навіть якщо хтось перехоплює інформацію в каналі зв’язку, він не зможе виявити узгоджений сеансовий ключ. KEM-алгоритми складаються з трьох основних компонентів, а загальний принцип їх роботи ілюструється на рисунку 2.1.

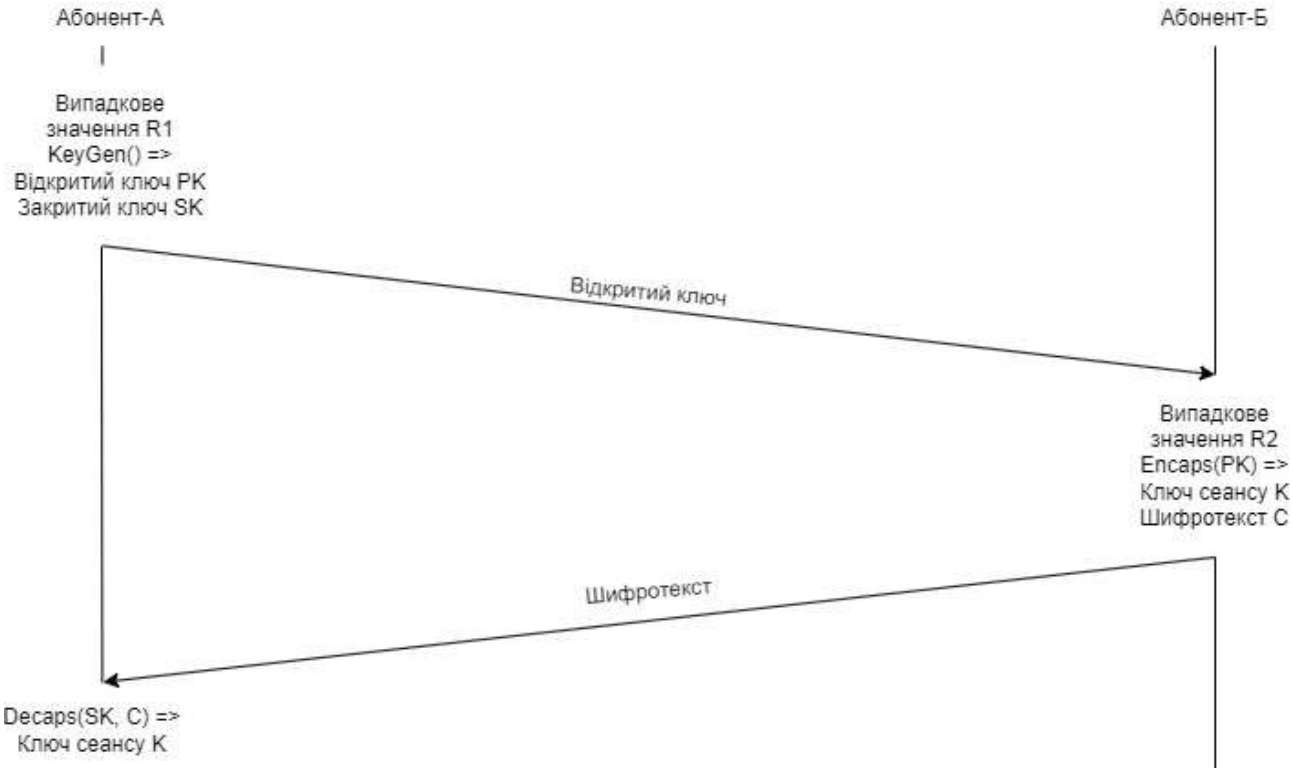

Рисунок 2.1 – Загальний принцип роботи KEM-алгоритмів

Крок 1: Операція KeyGen() генерує пару ключів – відкритий і закритий («PK/SK») – на основі випадкового вхідного значення («R1»). Закритий ключ зберігається у Абонента-А, тоді як відкритий ключ передається Абоненту-Б у незахищеному вигляді через мережу.

Крок 2: операція Encaps(PK) – дозволяє Абоненту-Б отримати сеансовий ключ («K») та випадковий шифротекст («C»), використовуючи відкритий ключ

(«PK») Абонента-А та випадкове вхідне значення («R2»). Ця операція забезпечує узгодження сеансового ключа для однієї сторони та створює необхідні вхідні дані для другої. Абонент-Б надсилає згенерований шифротекст Абоненту-А.

Крок 3: операція Decaps(SK, C) – дозволяє Абоненту-А отримати сеансовий ключ («K») за допомогою шифротексту («C»), згенерованого Абонентом-Б за допомогою відкритого ключа («PK»). Для цього потрібно виконати операцію Decaps(SK, C), яка приймає закритий ключ і шифротекст, в результаті чого генерується сеансовий ключ, що збігається з тим, який отримав Абонент-Б на 2-му кроці.

Крок 4: Тепер абоненти можуть переходити на симетричне шифрування, використовуючи узгоджений ключ сеансу, що забезпечує захищене з’єднання.
Порівняння алгоритмів для вибору найбільш підходящого варіанту реалізації у власному протоколі є важливим етапом аналізу. Для цього розглянемо алгоритми, обговорені в пунктах 2.2.1–2.2.5. Ключовим фактором є швидкодія, оскільки постквантова криптографія, як правило, працює повільніше за традиційну через більші розміри ключів та складніші обчислення. Тому для вибору оптимального рішення серед розглянутих постквантових алгоритмів необхідно визначити критерії їх порівняння.

Довжина відкритого та закритого ключів (в байтах) є важливим показником, що відображає складність алгоритму, оскільки вона безпосередньо впливає на кількість операцій, необхідних для знаходження ключа (у найгіршому випадку, для ключа довжиною n біт потрібно виконати 2^n операцій). Крім того, ця інформація вказує на обсяг даних, які потрібно передати через мережу (для відкритого ключа), а також впливає на час, необхідний для генерації пари ключів.

Узагальнений коефіцієнт швидкодії (0-10) – оцінка швидкодії алгоритму, яка представляє собою комбінацію попередніх трьох параметрів (генерація пари ключів, операції Encaps та Decaps) і вираховується за наступною формулою:

Нижня межа – 0, верхня – 10, так як кожен окремий критерій алгоритму демонструє максимальну швидкість не перевищуючи 100.000 операцій за секунду, то окрема частина коефіцієнту швидкодії вимірюється у межах 0-10, загальний коефіцієн – як сумма частин, поділенний на кількість цих частин. Результат підрахунку коефіцієнта округлений до сотих.

Таблиця 2.1 Порівняльні характеристики KEM-алгоритмів

Алгоритм і рівень безпеки NIST	Відкритий ключ (байт)	Закритий ключ (байт)	Генерація пари ключів (од/с)	Encaps (од/с)	Decaps (од/с)	Узагальнени й коефіцієнт швидкодії (0-10)
CRYSTALS -KYBER (5)	1568	3168	71094	61995	96649.33	7.66
NTRU (5)	1230	1590	2437.33	47725.33	44142.33	3.14
SABER (5)	1312	3040	38432.1	30412.7	31734.6	3.35
FrodoKEM (5)	21520	43088	1025.22	773.33	795.07	0.09
Classic McEliece (5)	1047319	13908	2.28	19568	6900	0.88
BIKE (5)	5122	16494	559.48	4996.67	227.11	0.19
HQC (5)	7245	7258	5566	2982.67	1792.67	0.34
SIKE (5)	564	48	104.79	50.17	127.4	0.01

Згідно з обраними критеріями оцінки, було проведено порівняльний аналіз, результати якого представлені в таблиці 2.1. Для цього порівняння використано алгоритми, що забезпечують найвищий доступний рівень безпеки NIST. У випадку, якщо кілька модифікацій алгоритму відповідають цьому рівню захисту, було обрано найкращий варіант.

На основі проведеного порівняння можна стверджувати, що кращим кандидатом є CRYSTAL-KYBER, оскільки він демонструє найвищу швидкодію та прийнятну довжину ключа. Якщо ж виникає потреба у впровадженні

альтернативних алгоритмів, варто розглянути SABER та NTRU, які показують добрі результати: SABER має високу швидкодію, а NTRU – оптимальний розмір ключа (не враховуючи SIKE через його компрометацію) та середню швидкодію. При виборі алгоритмів, що не ґрунтуються на ґратках, рекомендується обрати HQC або BIKE. Алгоритм Classic McEliece демонструє середню швидкодію, але має надзвичайно великі ключі (довжина публічного ключа перевищує один мільйон байт), що створює додаткове навантаження на канал зв'язку та час генерації пари ключів, хоча не забезпечує значних переваг, окрім захисту від атак методом перебору ключів.

Згідно з порівнянням у таблиці 2.1, найкращим варіантом для подальшої розробки є CRYSTAL-KYBER. Тому його буде реалізовано в протоколі в першу чергу як KEM-алгоритм.

Алгоритм автентифікації має підтвердити ідентичність однієї зі сторін зв'язку. Одним із можливих рішень є використання SSL-сертифікатів. Проте цей підхід передбачає залучення додаткової сторони – центру сертифікації, до якого клієнт повинен звертатися, використовуючи сертифікат, отриманий від сервера. Це додає ще один етап у процес встановлення з'єднання та вимагає додаткової конфігурації, як для отримання сертифікату сервером, так і для взаємодії клієнта з центром сертифікації.

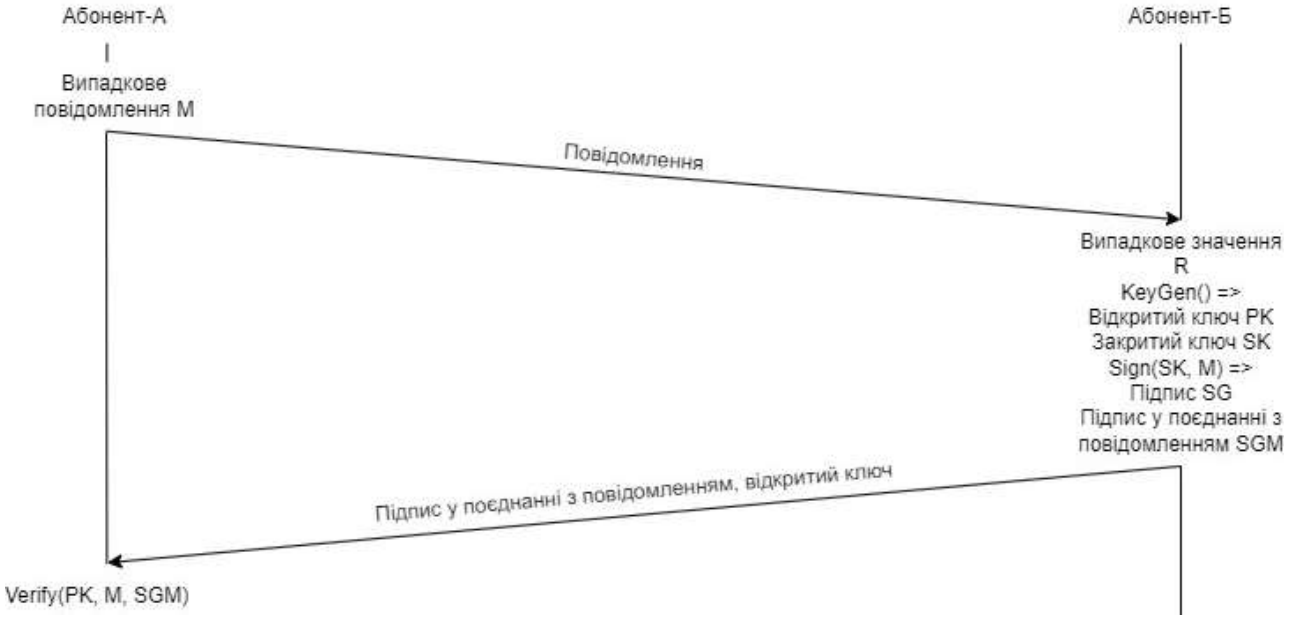

Рисунок 2.2 – Загальний принцип роботи алгоритму цифрового підпису

Алгоритм цифрового підпису може також використовуватися для автентифікації учасників під час обміну повідомленнями або даними в мережі. Враховуючи, що раніше було розглянуто кілька квантовостійких алгоритмів цифрового підпису, їх застосування є виправданим. Для автентифікації слід виконати наступні дії (загальний принцип роботи алгоритму цифрового підпису представлено на рисунку 2.2).

Крок 1: Абонент-А створює випадкове повідомлення («M») довільної довжини та надсилає його в незахищеному вигляді абоненту-Б через мережу.

Крок 2: Абонент-Б генерує пару ключів – відкритий і закритий («PK/SK») – за допомогою випадкового значення («R»). Потім, використовуючи закритий ключ («SK») та отримане повідомлення («M»), він створює цифровий підпис («SG»«). Абонент-Б передає згенерований підпис («SG») разом з повідомленням («M») у формі «SGM», а також відкритий ключ («PK»).

Крок 3: абонент-А використовує отриманий відкритий ключ («PK»), цифровий підпис («SG») та повідомлення («M») для перевірки, чи дійсно інша сторона підписала це повідомлення і чи є підпис дійсним. Для цього застосовується процедура перевірки підпису, яка гарантує автентичність сторін.

У процесі використання протоколу випадкове повідомлення («M») буде передано у незахищеному вигляді. Натомість відповідь від Абонента-Б буде зашифрована, використовуючи ключ сеансу, узгоджений за допомогою обраного KEM-алгоритму. Це підвищує безпеку, ускладнює доступ до публічного ключа, що застосовується алгоритмом, і зменшує кількість кроків, необхідних для встановлення з’єднання.

Порівняємо раніше розглянуті алгоритми цифрового підпису, щоб обрати найбільш підходящий варіант для реалізації в протоколі. Як і в випадку з алгоритмами отримання спільного секретного ключа, потрібно визначити критерії для проведення порівняльного аналізу. Загальні критерії, такі як розмір відкритих і закритих ключів, а також швидкість їх генерації, залишаються

актуальними. Крім того, вводяться специфічні критерії, що стосуються саме цього типу алгоритмів.

Sign (операцій на секунду) – це кількість підписувальних операцій, які алгоритм здатен виконати за одну секунду, використовуючи закритий ключ та випадкове повідомлення. Цей показник відображає частину загальної швидкості алгоритму.

Verify (операцій на секунду) – це кількість підписів, які алгоритм може перевірити за одну секунду, використовуючи відкритий ключ та цифровий підпис для отримання повідомлення. Цей показник також відображає частину загальної швидкості алгоритму.

Узагальнений коефіцієнт швидкодії (0-10) – оцінка швидкодії алгоритму, яка представляє собою комбінацію попередніх трьох спецефічних параметрів (генерація пари ключів, операції Sign та Verify) і вираховується за наступною формулою:

Нижня та верхня межі коефіцієнту коефіцієнту залишаються в діапазоні від 0 до 10 як і для KEM-алгоритмів (максимальна сума окремих показників знаходиться в межах 60.000, тому саме 60.000 – прийнято за еталоне значення – 10). Результат підрахунку коефіцієнта округлений до сотих.

Таблиця 2.2 Порівняльні характеристики DS-алгоритмів

Алгоритм і рівень безпеки NIST	Відкритий ключ (байт)	Закритий ключ (байт)	Генерація пари ключів (од/с)	Sign (од/с)	Verify (од/с)	Узагальнений коефіцієнт швидкодії (0-10)
CRYSTALS-DILITHIUM (5)	2592	4864	21586.33	9008.67	21314.33	8.65
FALCON (5)	1793	2305	42.09	1542.82	8759.33	1.72
Rainbow (5)	536136	1408736	0.18	23.82	24.48	0.01
GeMSS (5)	352180	32				
SPHINCS+(5)	64	128	955.67	44.22	1815	0.47

У цій категорії найкращим варіантом є CRYSTALS-DILITHIUM, алгоритм з прийнятною довжиною ключа, який демонструє найвищу продуктивність. Як альтернатива можна розглянути FALCON, який має коротший ключ, але помітно гіршу продуктивність, а також SPHINCS+, що відрізняється найменшою парою ключів і великою кількістю реалізацій, проте є найповільнішим з трьох алгоритмів за швидкістю операцій підписування та перевірки підпису. GeMSS наразі не реалізовано в проекті OQS, тому точні дані про його продуктивність відсутні, хоча за теоретичними оцінками він повинен демонструвати прийнятну швидкодію, подібну до алгоритму FALCON.

На базі порівняння з таблиці 2.2, кращим варіантом для подальшого використання є CRYSTAL-DILITHIUM, тому саме його буде реалізовано у протоколі для автентифікації сторін.

Після узгодження спільного закритого ключа та автентифікації сторін з’єднання, клієнт і сервер переходять до використання симетричного шифрування для захисту даних під час взаємодії. Це також може відбуватися раніше, щоб забезпечити конфіденційність відкритого ключа та підпису, необхідних для автентифікації. Симетричне шифрування є захищеним від атак як звичайних, так і квантових комп’ютерів, за умови використання достатньо великого ключа. Існує дві підкатегорії симетричного шифрування: блокове та потокове.

Обидві сторони застосовують один спільний ключ для шифрування та дешифрування повідомлень. Основною вразливістю цього підходу є використання єдиного ключа, оскільки його компрометація призводить до втрати конфіденційності зв’язку. Загальний принцип роботи симетричного алгоритму ілюструється на рисунку 2.3.

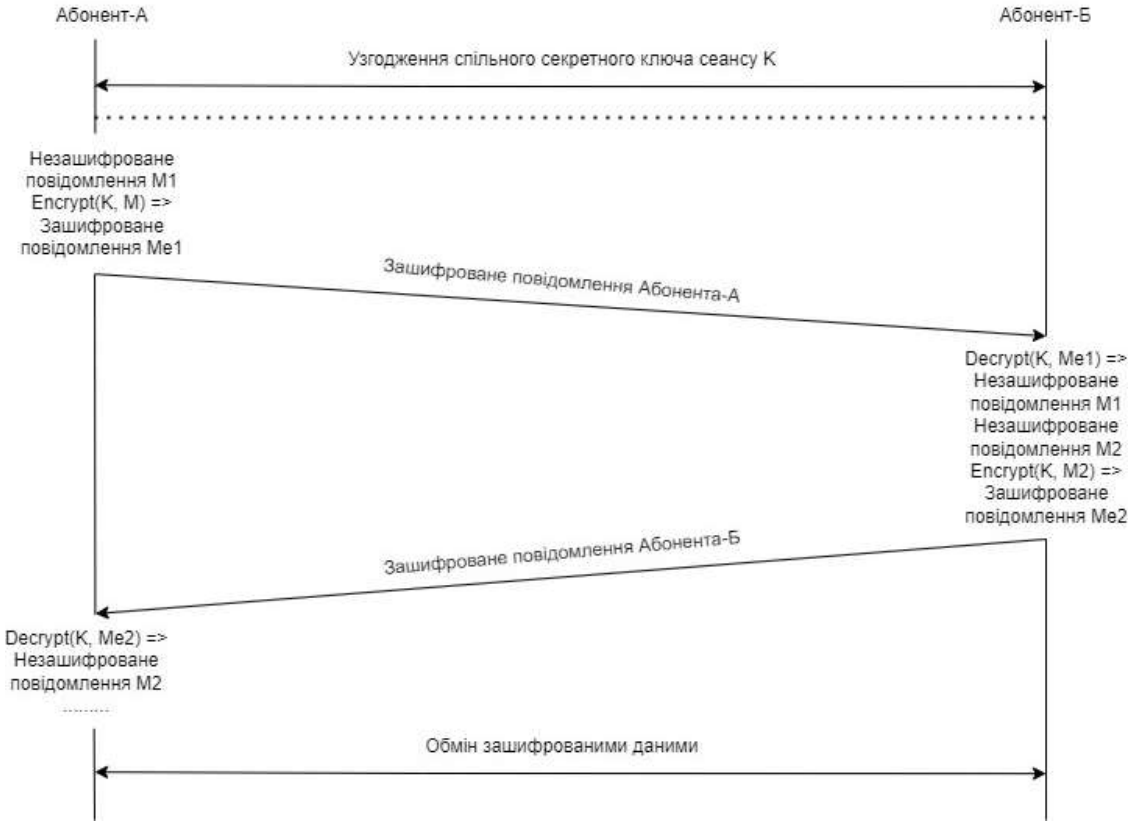

Рисунок 2.3 – Загальний принцип роботи симетричного алгоритму

Вибір симетричного алгоритму визначається конкретними завданнями та потребами користувача. При його виборі важливо враховувати такі критерії:

- Розмір ключа (біт): Довший ключ забезпечує вищий рівень безпеки, але може знижувати продуктивність і ускладнювати управління ключами.

- Розмір блоку (біт): Збільшення розміру блоку дозволяє швидше шифрувати великі обсяги даних, проте може призвести до зростання розміру шифротексту.

- Швидкодія (мс): Алгоритм повинен бути ефективним і економічним з точки зору обчислювальних ресурсів (результати наведені для випадкових вхідних даних обсягом 256 та 1024 байти).

- Використання пам’яті (Кб): Кількість пам’яті, яку алгоритм потребує для шифрування та дешифрування даних; чим менше використання пам’яті, тим краще (результати наведені для випадкових вхідних даних обсягом 256 та 1024 байти).

Згідно зі специфікацією NIST, 5-й рівень криптостійкості передбачає використання симетричного алгоритму з 256-бітним ключем. Відповідно, дані в таблиці 2.3 стосуються алгоритмів, що використовують ключі такого розміру. Завдання для вимірювання швидкодії полягає у створенні екземпляра класу, який реалізує відповідний алгоритм, а також у шифруванні та дешифруванні випадкового вхідного значення довжиною 256 та 1024 байти. Необхідно перевірити, чи збігається дешифроване значення з початковим.

Таблиця 2.3 Порівняльні характеристики симетричних алгоритмів

Алгоритм	Підтримуваний розмір ключа (біт)	Розмір блоку (біт)	Швидкодія (мс)	Використання пам’яті (Кб)
AES	128, 192 та 256	128	0.447 1.77	847.28 3348.9
ДСТУ 7624 (128)	128 та 256	128	0.456 1.801	847.41 3349.02
ДСТУ 7624 (256)	256 та 512	256	0.477 1.797	848.05 3349.67
Blowfish	32-448	64	0.48 1.847	849.49 3351.11
Twofish	128, 192 та 256	128	0.437 1.815	858.15 3359.77

За результатами порівняльного аналізу можна стверджувати, що доцільно використовувати AES (Advanced Encryption Standard) як один з найпопулярніших і найефективніших алгоритмів симетричного шифрування. AES забезпечує високий рівень безпеки при значній швидкості роботи та невеликому розмірі ключа (128, 192 або 256 біт). Додатково, розмір блоку в AES становить 128 біт, що дозволяє ефективно шифрувати великі обсяги даних і зменшує розмір шифротексту. AES та Twofish є одними з найшвидших алгоритмів у цій категорії.

AES є популярним стандартом, що має безліч наочних прикладів та реалізацій на різних мовах програмування. Цей алгоритм підтримується та оновлюється організацією NIST, а також має відкритий код, що дозволяє фахівцям з усього світу оцінювати та покращувати його безпеку. Важливо

зазначити, що AES рекомендований для використання в державному секторі США, що підтверджує його високу надійність та безпеку. Крім того, AES слугує еталоном для оцінки рівнів безпеки постквантових алгоритмів, що спрощує адаптацію до алгоритмів, які мають схожі характеристики.

Загалом, алгоритм симетричного шифрування має найменший вплив на зв'язність системи і може бути замінений на будь-який інший, наприклад, на алгоритм «Калина», затверджений національним стандартом України ДСТУ 7624:2014, без значних витрат часу.

Висновки до розділу 2

Під час дослідження математичних систем, що застосовуються в криптографії, було виявлено, що квантові алгоритми здатні розв'язувати проблеми, на яких ґрунтуються сучасні криптографічні алгоритми, що вважаються криптостійкими та надійними. Це створює загрозу для їх подальшого використання. Щоб зменшити ризики, пов'язані з впровадженням та використанням квантових систем, вже зараз необхідно планувати перехід на квантовостійкі технології.

Було проведено порівняльний аналіз квантовостійких криптографічних алгоритмів, які беруть участь у сертифікації від організації NIST. Ця організація займається розвитком постквантової криптографії через процес стандартизації та накопичує інформацію з проведених досліджень. Алгоритми, що розглядаються, базуються на математичних задачах, які, на відміну від традиційних криптографічних методів, вважаються складними для розв'язання як класичними, так і квантовими комп'ютерами. У процесі проектування та розробки найчастіше використовуються криптографічні методи на основі ґраток, хешування або кодів коригування помилок. Однак вибір конкретного рішення залежить від вимог до системи і має бути адаптований для кожного випадку окремо, враховуючи такі фактори, як обсяг пам'яті, швидкість обробки та довжина даних для передачі.

В результаті аналізу, проведеного в пункті 2.3, було ухвалено рішення щодо вибору конкретних алгоритмів. Зокрема, для реалізації у власному протоколі обрано алгоритми з сімейства CRYSTALS: CRYSTALS-KYBER для KEM та CRYSTALS-DILITHIUM для цифрового підпису. Ці алгоритми демонструють високу швидкодію, оптимальну довжину ключів і ґрунтуються на добре вивчених математичних задачах, що може забезпечити додаткову перевагу в контексті виявлення нових атак. Для симетричного шифрування даних у цій дипломній роботі обрано алгоритм AES, який буде використано в подальшій розробці та впровадженні.

Для підвищення безпеки протоколу важливо передбачити можливість його модифікації, застосовуючи алгоритми, що базуються на різних математичних моделях. Це допоможе зменшити ризик компрометації всього протоколу у випадку виявлення ефективної атаки на один із використовуваних алгоритмів. Найкращим рішенням у цьому випадку є дотримання принципу відкритості для модифікацій та закритості для змін (open/closed principle). Це означає, що код має бути написаний так, щоб компоненти можна було змінювати, при цьому загальна структура залишалася незмінною.

Крім того, важливо мати можливість використовувати відкриті реалізації обраних криптографічних протоколів на різних мовах програмування. Це допомагає зменшити потенційні ризики атак на вразливості реалізацій і забезпечує підтримку з боку спільноти. Зазвичай, такі атаки, якщо вони виникають і можуть бути виправлені в рамках конкретного алгоритму, мають найвищий пріоритет, і їх вирішенню приділяється максимальна увага з боку розробників та користувачів програмної реалізації алгоритму.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ АЛГОРИТМУ ШИФРУВАННЯ

3.1. Специфікація протоколу шифрування

Протокол доцільно створювати за аналогією з протоколом TLS, який функціонує на рівні представлення (6-й рівень за OSI-моделлю) або на прикладному рівні (за TCP/IP-моделлю). Такий підхід дозволяє інтегрувати його з високорівневими протоколами, такими як HTTP або протоколи електронної пошти, без потреби змінювати їхню реалізацію. Крім того, це забезпечує гнучкість у виборі транспортної основи – протокол може працювати як з сесійними інтерфейсами на основі сокетів, так і з новими рішеннями, побудованими безпосередньо на транспортному рівні, наприклад, TCP або UDP.

Одним із ключових аспектів проєктування власного протоколу є вибір щодо шифрування: чи варто обмежитися одним шифронабором, чи забезпечити підтримку декількох з можливістю узгодження між клієнтом і сервером. Підтримка кількох шифронаборів надає значну гнучкість – як для адаптації протоколу під різні потреби клієнтів, так і для його майбутньої модернізації без зміни основної структури, що відповідає сучасним принципам розробки. Водночас це створює потенційні ризики: якщо протокол дозволяє використовувати шифронабір зі скомпрометованим алгоритмом, зловмисники можуть цим скористатися, особливо якщо такий набір є пріоритетним за замовчуванням.

Оскільки навіть сьогоднішні квантостійкі алгоритми можуть втратити безпечність (прикладом є атака Magma на SIKE, згадана в пункті 2.2.5), підтримка декількох шифронаборів підвищує надійність і стійкість протоколу. Враховуючи ці переваги, майбутній протокол наслідуватиме підхід TLS і передбачатиме можливість вибору з кількох шифронаборів, які узгоджуються сторонами під час з'єднання. Відповідні вимоги до протоколу будуть сформульовані на основі попереднього аналізу.

Незалежність від транспортного та сесійного рівнів (у моделі OSI) або транспортного рівня (у TCP/IP) – протокол має бути незалежним від конкретної реалізації нижчих мережевих рівнів. Це дозволить уникнути прив’язки до певних технологій або платформ, забезпечуючи можливість самостійного та гнучкого розвитку протоколу в майбутньому.

Кросплатформеність – розроблений протокол повинен коректно працювати на різних операційних системах, зокрема Windows, Linux та macOS. Такий підхід розширює коло потенційних користувачів і полегшує інтеграцію протоколу в існуючі програмні рішення.

Стійкість до квантових атак – протокол має гарантувати безпеку як у випадку класичних атак, так і при використанні квантових обчислень та відповідних алгоритмів, що є важливим чинником довгострокової криптографічної стійкості.

Висока продуктивність – при встановленні захищеного з'єднання протокол має забезпечувати час ініціалізації не більше 200 мс (без врахування мережевих затримок), а також здатність обробляти не менше 150 Мб даних на секунду, що відповідає максимальній пропускній здатності гігабітного каналу.

Розроблюваний протокол матиме завдання забезпечити захист передаваних даних у незахищених каналах зв’язку, при цьому не впливаючи на функціонування протоколів вищих або нижчих рівнів. Наш протокол має використовувати квантостійкі криптографічні алгоритми та гарантувати безпечний канал обміну інформацією між кінцевими точками.

Реалізація звичайного режиму призначена для встановлення захищеного каналу зв’язку між клієнтом і сервером у два основні етапи, аналогічно до TLS 1.3, однак зменшує кількість обмінів даними під час фази встановлення з’єднання. У ньому використовуються:

KEM-алгоритм – для узгодження спільного сеансового ключа;

Алгоритм цифрового підпису – для автентифікації сервера;

Симетричний алгоритм шифрування – для захисту даних після встановлення ключа.

Процедура встановлення захищеного з’єднання у звичайному режимі передбачає три основні кроки (і до двох додаткових, якщо обраний клієнтом шифронабір не підтримується сервером). Весь процес включає лише один зворотний обмін повідомленнями, що дозволяє ефективно налаштувати безпечний канал зв’язку.

Крок 1. Попередня конфігурація:

Клієнт має список підтримуваних шифронаборів SCS1, з якого може опціонально вибрати пріоритетний – PCS1. Якщо вибір не зроблено, використовується перший елемент зі списку. Також клієнт встановлює режим з’єднання CM (для звичайного режиму – значення 0). Сервер своєю чергою має власний набір підтримуваних шифронаборів SCS2.

Крок 2. Повідомлення ClientInit:

Клієнт генерує випадкове число R1 і на основі обраного шифронабору PCS1 створює екземпляр KEM-алгоритму (PCS1.KEM). За допомогою функції KeyGen() генерується пара ключів – відкритий (KPK) і закритий (KSK). Потім формується повідомлення ClientInit, що включає обраний шифронабір PCS1, режим CM та згенерований відкритий ключ KPK. Повідомлення передається серверу у відкритому вигляді та містить механізми перевірки цілісності.

Крок 3. Повідомлення ServerInit:

Сервер приймає ClientInit, перевіряє його цілісність. У разі помилки – з’єднання припиняється. Далі сервер перевіряє, чи підтримується ним шифронабір, запропонований клієнтом (PCS1). Якщо так – продовжує обробку, інакше ініціюється процедура резервного узгодження.

Якщо сервер не підтримує запропонований клієнтом шифронабір, він повинен сформувати повідомлення UnsupportedClientParams, у якому буде вказано перелік підтримуваних шифронаборів (SCS2). Клієнт, у свою чергу, має вибрати один із цих шифронаборів (за умови, що підтримує хоча б один із них)

та повторно сформувати повідомлення ClientInit, вже використовуючи новий шифронабір – умовно позначений як PCS11.

Якщо ж сервер підтримує обраний клієнтом шифронабір, він погоджується на його використання і вважає його основним для поточного сеансу.

Після узгодження шифронабору сервер також погоджує режим роботи (CM), використовуючи передане клієнтом значення, за умови його коректності (для «звичайного режиму» воно має дорівнювати 0).

Далі, сервер генерує випадкове число R2 та створює власний екземпляр відповідного KEM-алгоритму (PCS2.KEM, де PCS2 – узгоджений шифронабір). За допомогою операції Encaps(KPK) та отриманого від клієнта публічного ключа (KPK), сервер генерує спільний сеансовий ключ K і шифротекст C. Клієнт зможе відтворити цей же ключ, використовуючи свій приватний ключ.

Після цього сервер ініціалізує алгоритм цифрового підпису, передбачений шифронабором PCS2 (PCS2.DSA). Використовуючи KeyGen(), він генерує пару ключів: відкритий (DPK) і закритий (DSK). Потім обчислює дайджест отриманого повідомлення ClientInit (позначимо його як SM) і підписує його, використовуючи функцію Sign(DSK, SM), що дає цифровий підпис SIG.

Для подальшої роботи із захищеним каналом, сервер ініціалізує симетричний шифрувальний алгоритм PCS2.SYM, використовуючи раніше згенерований ключ K. Цей алгоритм застосовується для шифрування всіх наступних даних у рамках з’єднання. Зокрема, з його допомогою сервер шифрує як підпис SIG, так і відкритий ключ DPK, отримуючи зашифроване повідомлення ESWK.

На завершальному етапі сервер формує повідомлення ServerInit, яке включає шифротекст C (отриманий під час Encaps) та шифровані дані ESWK, і надсилає його клієнту.

Крок 4. Клієнт отримує повідомлення ServerInit та перевіряє його цілісність. Далі, за допомогою функції Decaps(KSK, C) KEM-алгоритму

PCS1.KEM та отриманого шифротексту C, він обчислює спільний сеансовий ключ K.

Використовуючи узгоджений шифронабір PCS1 і ключ K, клієнт ініціалізує симетричний алгоритм шифрування PCS1.SYM. Окрема перевірка на коректність обчислення ключа відбувається під час розшифрування підпису, тому на цьому етапі помилки синхронізації ключів виникнути не повинні.

За допомогою ініціалізованого симетричного алгоритму клієнт розшифровує дані, що містять цифровий підпис SIG та відкритий ключ DPK алгоритму цифрового підпису (DS-алгоритму).

Після цього клієнт обчислює дайджест свого раніше надісланого повідомлення ClientInit (позначеного як SM) та створює екземпляр алгоритму цифрового підпису PCS1.DSA, використовуючи який він виконує перевірку підпису через операцію Verify(DPK, SM, SIG). Якщо перевірка не проходить – клієнт має завершити сеанс і повернути відповідну помилку.

Крок 5. Якщо всі параметри були успішно узгоджені й перевірені (сервер надіслав підпис, захищений симетричним шифруванням; клієнт його успішно розшифрував та автентифікував), вважається, що захищений канал зв’язку між клієнтом і сервером успішно встановлено.

3.2. Структура «клієнт-серверних» повідомлень

Після завершення розробки специфікації протоколу наступним етапом стане створення структури повідомлень, які забезпечуватимуть виконання всіх необхідних функцій. Основним пріоритетом під час проєктування повідомлень буде максимальне скорочення їх довжини з метою оптимізації обсягу переданих даних у мережі. Такий підхід дозволяє зменшити навантаження на канал зв’язку, хоч і ускладнює подальше розуміння та аналіз структури повідомлень.

Протокол передбачає підтримку таких типів повідомлень:

– ClientInit

– ServerInit

– UnsupportedClientParams
– ClientFinish
– DataTransfer
– Close

Кожне повідомлення починається з заголовка фіксованого формату, який є спільним для всіх типів повідомлень. Довжина заголовка складатиме 10 байтів і міститиме основну службову інформацію. Детальна структура заголовка буде представлена у таблиці 3.1

Таблиця 3.1 – Структура заголовку повідомлень, які використовуються в протоколі

№ байту	Призначення
0	Версія протоколу (0 – 255)
1	Тип повідомлення (0 – 255)
2 – 5	Довжина тіла повідомлення ($0 – 2^{32}$)
6 – 9	Частина обчисленного дайджесту повідомлення, використовується для перевірки цілосності (масив байт)

Наступним кроком стане проєктування структури кожного з підтримуваних повідомлень протоколу. Кожне повідомлення складається з заголовка та основної частини (тіла). Заголовок включає службову інформацію, необхідну для коректної обробки повідомлення в межах протоколу, тоді як тіло містить корисне навантаження, специфічне для кожного типу повідомлення.

ClientInit – це ініціалізуюче повідомлення, яке надсилається клієнтом на початку сеансу. Воно містить усю необхідну інформацію, що дозволяє серверу згенерувати сеансовий ключ, створити цифровий підпис (у разі потреби, залежно від конфігурації клієнта), а також сформувати відповідь – повідомлення ServerInit. Структура повідомлення ClientInit описана у таблиці 3.2.

Таблиця 3.2 – Стркутура повідомлення ClientInit:

№ байту	Призначення
0 – 9	Заголовок повідомлення
10	Ідентифікатор бажаного клієнтом для використання шифронабору (0 – 255)
11	Ідентифікатор обраного режиму роботи (0 – 2)
12 – кінець	Публічний ключ КЕМ-алгоритму (масив байт)

ServerInit – це повідомлення-відповідь, яке сервер надсилає у відповідь на ClientInit, за умови підтримки ним запропонованого клієнтом шифронабору. Повідомлення містить усю необхідну інформацію для клієнта, щоб завершити узгодження сеансового ключа, а також, за потреби, виконати автентифікацію сервера.

Його структура повинна відповідати формату, наведеному у таблиці 3.3. Повідомлення також може містити додаткові (опціональні) поля, наявність яких залежить від обраного режиму роботи протоколу. Зокрема, у «звичайному режимі» ці параметри мають бути присутні, тоді як в інших режимах повідомлення формується без них.

Таблиця 3.3 – Стркутура повідомлення ServerInit

№ байту	Опціональна частина	Призначення
0 – 9	Ні	Заголовок повідомлення
10 – 13	Ні	Довжина шифротексту (ДШ) (0 – 2^{32})
14 – 14+ДШ	Ні	Шифротекст (масив байт)
15+ДШ – 18+ДШ	Так	Довжина цифрового підпису (0 – 2^{32})
19+ДШ – 22+ДШ	Так	Довжина публічного ключа DS- алгоритму (0 – 2^{32})
23 – кінець	Так	Зашифроване поєднання цифрового підпису у поєднанні з публічним ключем DS-алгоритму (масив байт)

UnsupportedClientParams – це повідомлення, яке сервер надсилає клієнту в тому випадку, якщо не підтримує запропонований ним шифронабір. У повідомленні має міститися інформація про всі шифронабори, які підтримує сервер, у вигляді 64-бітної бітової маски. Важливо, щоб і клієнт, і сервер використовували однакові позиції бітів для відповідних алгоритмів, аби забезпечити правильне кодування та декодування маски. Структура цього повідомлення описана у таблиці 3.4.

Таблиця 3.4 – Стркутура повідомлення UnsupportedClientParams

№ байту	Призначення
0 – 9	Заголовок повідомлення
10 – 17	Набір пітримуваних сервером шифронаборів (64-бітна маска)

ClientFinish – додаткове повідомлення, яке використовується лише у випадку роботи протоколу в «швидкому режимі». Воно потрібне для додаткової перевірки сервером коректності встановлення параметрів з’єднання. Структуру повідомлення буде представлено у таблиці 3.5.

Таблиця 3.5 – Стркутура повідомлення ClientFinish

№ байту	Призначення
0 – 9	Заголовок повідомлення
10 – кінець	Зашифрований симетричним алгоритмом дайджест повідомлення ServerInit (масив байт)

DataTransfer – повідомлення, що використовується під час основного етапу обміну даними між сторонами після успішного встановлення захищеного з’єднання. Його мета – передача зашифрованих даних з використанням симетричного алгоритму, обраного в межах узгодженого шифронабору. Повідомлення складається з заголовка та тіла, де тіло – це зашифрований блок даних, який клієнт або сервер хоче передати по мережі. Формат структури подано у таблиці 3.6.

Таблиця 3.6 – Стркутура повідомлення DataTransfer

№ байту	Призначення
0 – 9	Заголовок повідомлення
10 – кінець	Зашифроване симетричним алгоритмом повідомлення

Close – повідомлення, призначене для коректного завершення з'єднання між клієнтом і сервером. Воно забезпечує звільнення виділених ресурсів, видалення узгодженої протоколом інформації та інші завершальні дії. Повідомлення не містить тіла і складається лише із заголовка. Перевірка цілісності для цього повідомлення не передбачена. Структура повідомлення Close наведена у таблиці 3.7.

Таблиця 3.7 – Стркутура повідомлення Close

№ байту	Призначення
0	Версія протоколу (0 – 255)
1	Тип повідомлення (6)
2 – 5	Довжина тіла повідомлення (0 0 0 0)
6 – 9	Частина обчисленного дайджесту повідомлення, використовується для перевірки цілосності (0 0 0 0)

3.3 Вибір інструментів та архітектри протоколу

Для реалізації протоколу було обрано мову програмування C# та платформу .NET. Зокрема, використання універсальної платформи .NET 7 [39] було визначено як оптимальне рішення, оскільки це високорівневий фреймворк, що підтримує крос-платформність і забезпечує прийнятну швидкодію завдяки JIT-компіляції. Крім того, платформа має широкий набір готових бібліотек і застосунків. Варто зауважити, що хоча .NET 7 є мажорним релізом, він має короткострокову підтримку (англ. short time support) тривалістю 18 місяців. Однак через мінімальну кількість залежностей оновлення до наступної версії, коли вона з’явиться, не має бути складним або трудомістким.

Щодо залежностей – сторонні бібліотеки не використовуються, застосовуються лише стандартні компоненти, що входять до складу фреймворку. Зокрема, System.Security.Cryptography надає доступ до широкого спектру криптографічних сервісів, включаючи симетричні алгоритми шифрування (Aes та AesGcm) і хеш-функції (SHA384), які використовуються у протоколі. Модуль System.Net.Sockets забезпечує кросплатформену роботу з мережевими сокетами. Оскільки протокол покликаний підтримувати будь-які мережеві примітиви для встановлення захищеного каналу зв’язку, у прикладі реалізовано базове «небезпечне» з’єднання на основі сокетів, що відзначається простотою та високою продуктивністю.

Водночас слід враховувати, що користувачі протоколу повинні мати встановлений .NET 7 на своїх пристроях. Після завершення підтримки цієї версії буде необхідно оновитися до новішої для забезпечення безпеки і сумісності. Загалом, використання .NET 7 відкриває багато переваг і значно спрощує розробку та експлуатацію програми.

Програмна реалізація протоколу складається з трьох основних проєктів, а також одного тестового проекту (Додаток А). Основні проєкти виконані у вигляді бібліотек класів – набору абстракцій і конкретних реалізацій, що легко піддаються повторному використанню, а також містять консольний додаток для демонстрації роботи протоколу. Загальна структура розробленої системи показана на рисунку 3.1.

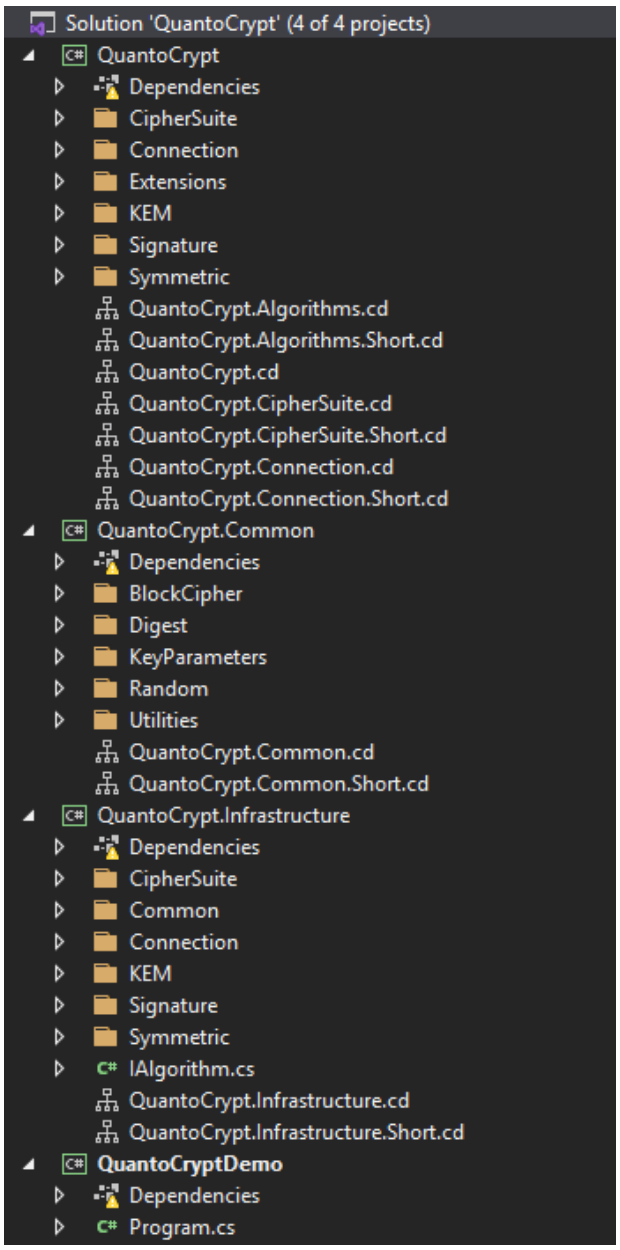

Рис. 3.1 Загальна структура розроблюваної програмної реалізації

Кожен проєкт має власну специфічну функцію, що дозволяє розділити загальну логіку на окремі, взаємопов’язані модулі. Такий підхід значно полегшує тестування, сприяє подальшому розширенню функціоналу протоколу та знижує зв’язність коду.

3.3. Результати роботи програмної реалізації протоколу шифрування

За підсумками розробки створено прототип мережевого протоколу, який забезпечує встановлення безпечного квантостійкого каналу зв’язку. Реалізація виконана у вигляді бібліотек на базі фреймворку .NET, що відкриває широкі

можливості для його застосування та інтеграції як у нові, так і в існуючі програмні продукти.

Метою створення прототипу було перевірити його життєздатність. Особлива увага приділялася швидкодії, оскільки порівняно з традиційною криптографією, квантостійкі протоколи потребують більше часу і обчислювальних ресурсів. Отримані дані дозволяють оцінити ефективність рішення та порівняти його з поширеними протоколами, такими як TLS. Результати засвідчують необхідність подальшої оптимізації та підвищення продуктивності протоколу.

Для перевірки функціональності розробленого протоколу було створено консольний додаток з попередньо налаштованими прикладами, які демонструють встановлення безпечного з’єднання між сервером і клієнтом з використанням різних режимів роботи протоколу. Також передбачено приклад обробки ситуації, коли клієнт намагається використати непідтримуваний сервером шифронабір. Це дозволяє кінцевим користувачам та розробникам краще зрозуміти процес встановлення з’єднання, оскільки додаток виводить інформацію про надіслані та отримані повідомлення у консоль (або інше налаштоване місце). Приклад роботи демонстраційного додатку наведено на рисунку 3.2, де показано інформацію про відправлене повідомлення ClientInit та початок повідомлення ServerInit, яке надіслав сервер.

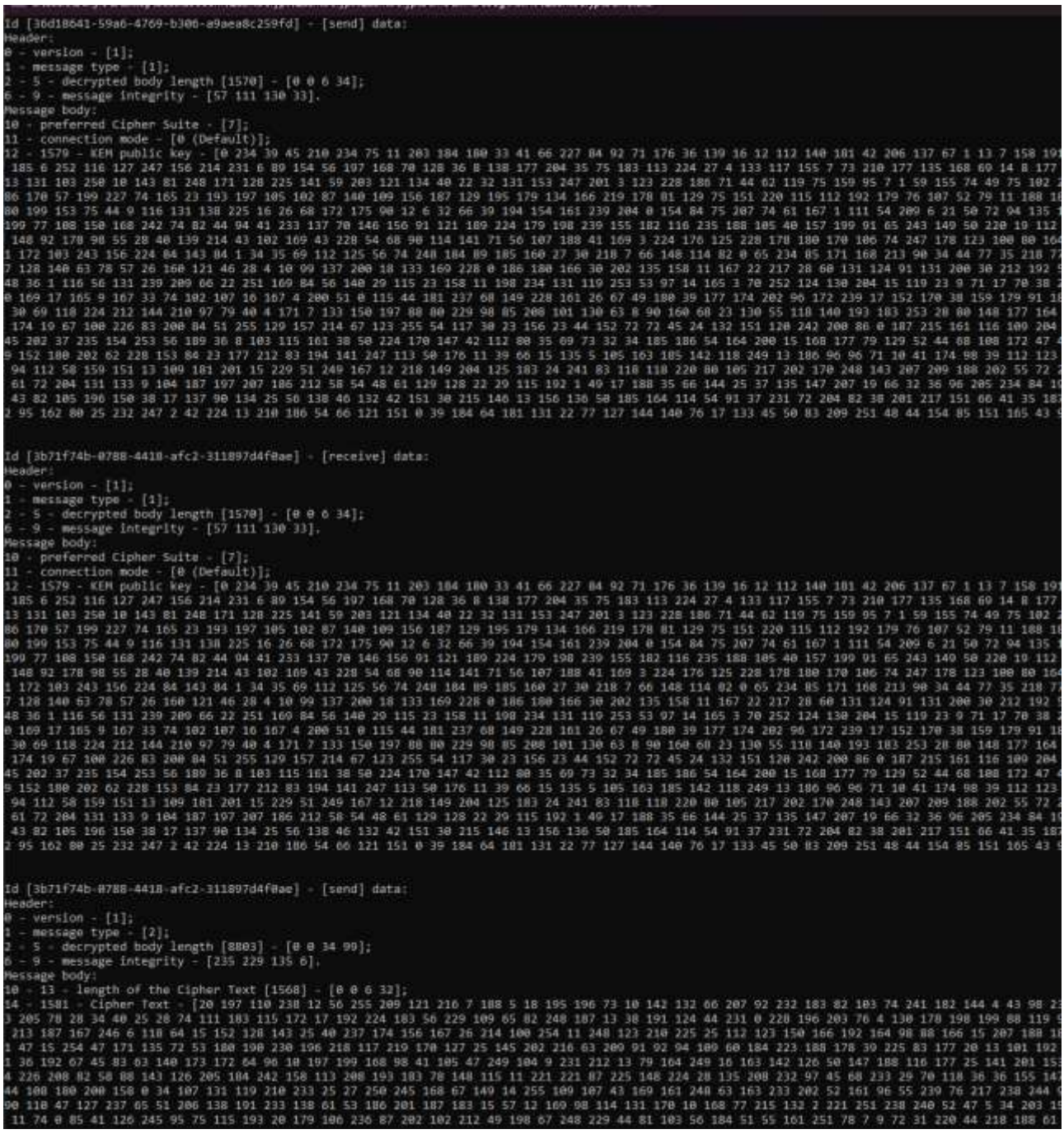

Рис. 3.2 Результати тестування протоколу

Висновки до Розділу 3

У цьому розділі сформульовано основні вимоги до протоколу, зокрема: можливість розширення набору підтримуваних алгоритмів і шифронаборів; незалежність від транспортного та сесійного рівнів згідно з моделлю OSI; кросплатформність; квантову стійкість; а також забезпечення прийнятної швидкодії.

При розробці структури повідомлень основним завданням було уніфікувати їх формат та максимально зменшити обсяг передаваних даних. Наприклад, для передачі інформації про підтримувані шифронабори використано бітову маску замість окремих ідентифікаторів чи назв, що значно зменшує обсяг інформації – 8 байт дозволяють описати до 64 шифронаборів. Кожне повідомлення містить заголовок із типом повідомлення та іншим необхідним метаданими, тоді як тіло повідомлення включає корисне навантаження, яке варіюється залежно від типу повідомлення та параметрів з’єднання.

Також наведено опис інструментів розробки та детально розглянуто архітектуру протоколу, що може бути реалізована на різних об’єктно-орієнтованих мовах програмування. Щодо програмної реалізації, описано базові абстракції, з якими працює протокол, представлено загальну структуру та розподіл компонентів по бібліотеках, а також детально проаналізовано основні розроблені модулі системи. Окремо виділено переваги і недоліки прийнятих рішень та наведено можливості для подальшого розвитку протоколу.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено мережевий протокол, який забезпечує створення захищеного зашифрованого каналу зв’язку поверх будь-якого незахищеного мережевого клієнт-сереверного з’єднання. Для цього застосовані шифронабори, що дозволяють протоколу використовувати необхідні криптографічні примітиви, зокрема алгоритми генерації спільного секретного ключа, автентифікації та симетричного шифрування даних. Використання квантово-стійких алгоритмів для цих примітивів гарантує захист від атак, що можуть бути реалізовані як класичними, так і квантовими комп’ютерами та технологіями.

Тестування зазначеного протоколу дозволило випробувати раніше досліджені теоретичні концепції на практиці та виявити їх сильні та слабкі сторони і окреслити особливості застосування. Отримані результати дозволили сформувати уявлення про напрямки застосування, подальшого вдосконалення розробленої системи. Зокрема:

- слід розширити підтримку мережевих каналів та криптографічних алгоритмів для подальшого розвитку сфери використання протоколу.

- доцільно додати можливість вибору алгоритму перевірки цілісності даних, адже наразі використовується лише SHA-384. Це дасть користувачам змогу налаштовувати протокол відповідно до своїх потреб.

- вартує покращити продуктивність (особливо при додаванні нових, більш повільних алгоритмів, оскільки для прототипу обрано найшвидші варіанти). Для чого можна розглянути використання швидших мов програмування, наприклад C++, а також замінити виклики в протоколі на оптимізовані реалізації.

- можна розглянути застосування більш продуктивних алгоритмів квантової криптографії або алгоритмів з нижчим рівнем безпеки які будуть працювати швидше (що важливо для локальних систем, де рівень кіберзагроз є меншим).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Полторак В.П. Криптографічний захист даних в цифрових інформаційних системах (Частина 1) / В.П. Полторак // Телеком. Військовий зв'язок. hi- Tech.ua. Спеціальний випуск. – Київ: Издательский дом СофтПресс, 2018. – №2.- С. 98-104;

2. Полторак В.П. Теорія інформації та кодування: Підручник / Ю.П. Жураковський, В.П. Полторак. – К.: Вища школа, 2001. – 255 с;

3. RFC Editor [Електронний ресурс] – Режим доступу до ресурсу: https://www.rfc- editor.org/;

4. SSL Pulse – continuous and global dashboard for monitoring the quality of SSL/TLS support [Електронний ресурс] – Режим доступу до ресурсу: https://www.ssllabs.com/ssl-pulse/;

5. Ahmad N. Analysis of Network Security Threats and Vulnerabilities by Development & Implementation of a Security Network Monitoring Solution [Електронний ресурс] / N. Ahmad, M. Habib. – 2010. – Режим доступу до ресурсу: https://www.researchgate.net/publication/202784990 Analysis of Network Sec urity Threats and Vulnerabilities by Development Implementation of a Secu rity Network Monitoring Solution;

6. Paterson K. Attacking the IPsec Standards in Encryption-only Configurations [Електронний ресурс] / K. Paterson, J. Degabriele // IEEE Xplore. – 2007. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/4223237;

7. Гапак О. М. Криптоаналіз. Криптографічні протоколи [Електронний ресурс] / О. М. Гапак, М. І. Глебена, П. П. Горват. – 2021. – Режим доступу до ресурсу: https : //www.uzhnu.edu.ua/uk/infocentre/get/36273 ;

8. Preston R. Applying Grover's Algorithm to Hash Functions: A Software Perspective / Richard H Preston. // IEEE Transactions on Quantum Engineering. – 2022. – №3. – С. 1-10;

9. Brute-Force Cryptanalysis with Aging Hardware: Controlling Half the Output of SHA-256 [Електронний ресурс] / M.Bouam, C. Bouillaguet, C. Delaplace, C. Noûs // HAL open science. – 2021. – Режим доступу до ресурсу:

https://hal.science/hal-02306904v2;

10. Полторак В. Технології постквантової криптографії / В. Полторак, Д. Новиков. // Адаптивні системи автоматичного управління. – 2023. – №1(42);

11. NIST (National Institute of Standards and Technology) [Електронний ресурс] – Режим доступу до ресурсу: https://www.nist. gov/;

12. Post-Quantum Lattice-Based Cryptography Implementations: A Survey / [H. Nejatollahl, N. Dutt, S. Ray та ін.]. // ACM Computing Surveys. – 2019. – №51. – С. 1-41;

13. Aggarwal D. Improved algorithms for the Shortest Vector Problem and the Closest Vector Problem ш the mfirnty norm [Електронний ресурс] / D. Aggarwal, P. Mukhopadhyay // arxiv. – 2018. – Режим доступу до ресурсу: https://doi.org/10.48550/arXiv.1801.02358;

14. Voulgaris P. Algorithms For The Closest And ShortestVector Problems On General Lattices [Електронний ресурс] / Panagtotis Voulgaris // UC San Diego Electronic Theses and Dissertations. – 2011. – Режим доступу до ресурсу: https://escholarship.org/uc/item/4zt7x45z;

15. Towards Classical Hardness of Module-LWE: The Linear Rank Case [Електронний ресурс] / K.Boudgoust, C. Jeudy, A. Roux-Langlois, W. Wen // Springer. – 2020. – Режим доступу до ресурсу: https://doi.org/10.1007/978-3- 030-64834-3 10;

16. Lyubashevsky V. A Toolkit for Ring-LWE Cryptography [Електронний ресурс] / V. Lyubashevsky, C. Peikert, O. Regev // Springer. – 2013. – Режим доступу до ресурсу: https://doi.org/10.1007/978-3-642-38348-9 3;

17. Tweaking the Asymmetry of Asymmetric-Key Cryptography on Lattices: KEMs and Signatures of Smaller Sizes [Електронний ресурс] / [J. Zhang, Y. Yu, S. Fan та ін.] // Springer. – 2020. – Режим доступу до ресурсу: https://doi.org/10.1007/978-3-030-45388-6 2;

18. Ding J. A Key Exchange Based on the Short Integer Solution Problem and the Learning with Errors Problem [Електронний ресурс] / J. Ding, K. Schmitt, Z. Zhang // Springer. – 2019. – Режим доступу до ресурсу: https://doi.org/10.1007/978-3-030-16458-4 8;

19. Post-Quantum Cryptography Project – Selected Algorithms 2022 [Електронний ресурс] – Режим доступу до ресурсу: https://csrc.nist.gov/Projects/post- quantum-cryptography/selected-algorithms-2022;

20. Garcia-Escartin J. Quantum collision finding for homomorphic hash functions [Електронний ресурс] / J. Garcia-Escartin, V. Gimeno, J. Moyano-Fernandez. – 2021. – Режим доступу до ресурсу: https://doi.org/10.48550/arXiv.2108.00100;

21. Practical Fault Injection Attacks on SPHINCS [Електронний ресурс] / A.Genet, M. Kannwischer, H. Pelletier, A. McLauchlan // Cryptology ePrint Archive. – 2018. – Режим доступу до ресурсу: https://eprint.iacr.org/2018/674;

22. Haraka v2 – Efficient Short-Input Hashing for Post-Quantum Applications / S.Kölbl, M. Lauridsen, F. Mendel, C. Rechberger. // IACR Transactions on Symmetric Cryptology. – 2016. – №2. – С. 1-29;

23. Classic McEliece [Електронний ресурс] – Режим доступу до ресурсу: https://classic.mceliece.org/;

24. Post-Quantum and Code-Based Cryptography–Some Prospective Research Directions [Електронний ресурс] / C.Balamurugan, K. Singh, G. Ganesan, M. Rajarajan // Cryptography. – 2021. – Режим доступу до ресурсу: https://doi.org/10.3390/cryptography5040038;

25. Post-Quantum Cryptography Project – Round 4 Submissions [Електронний ресурс] – Режим доступу до ресурсу: https://csrc.nist.gov/Projects/post- quantum-cryptography/round-4-submissions;

26. Galbraith S. Computational problems in supersingular elliptic curve isogenies / S. Galbraith, F. Vercauteren. // Quantum Inf Process. – 2018. – №17;

27. De Feo L. Mathematics of Isogeny Based Cryptography [Електронний ресурс] / Luca De Feo // arxiv. – 2017. – Режим доступу до ресурсу: https://doi.org/10.48550/arXiv.1711.04062;

28. SIKE [Електронний ресурс] – Режим доступу до ресурсу: https://sike.org/;

29. Overview of NIST Round 3 Post-Quantum cryptography Candidates [Електронний ресурс]. – 2020. – Режим доступу до ресурсу:

https://www.pqsecurity.com/wp-content/uploads/2020/07/Round-3.pdf;

30. Castryck W. An efficient key recovery attack on SIDH [Електронний ресурс] / W. Castryck, T. Decru // Cryptology ePrint Archive. – 2022. – Режим доступу до ресурсу: https://ia.cr/2022/975;

31. Інформаційні технології. Криптографічний захист інфомації. Цифровий підпис, що грунтується на еліптичних кривих: ДСТУ 4145:2002. – Держспоживстандарт Україні, 2002. – 36 с.;

32. The IBM Quantum Development Roadmap [Електронний ресурс] // IBM – Режим доступу до ресурсу: https: //www.ibm.com/quantum/roadmap;

33. Open Quantum Safe (OQS) project [Електронний ресурс] – Режим доступу до ресурсу: https: //openquantumsafe.org/;

34. Вихідний код бібліотеки liboqs [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/open-quantum-safe/liboqs;

35. OQS algorithm performance visualizations [Електронний ресурс] – Режим доступу до ресурсу: https: //openquantumsafe. org/benchmarking/;

36. NIST – Cryptographic Standards and Guidelines – AES Development [Електронний ресурс] – Режим доступу до ресурсу: https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development;

37. Barker E. Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms [Електронний ресурс] / Elaine Barker // NIST. – 2020. – Режим доступу до ресурсу: https://doi.org/10.6028/NIST.SP.800-175Br1;

38. Інформаційні технології. Криптографічний захист інфомації. Алгоритм симетричного блокового перетворення: ДСТУ 7624:2014. – Держспоживстандарт України, 2014. – 238 с.

39. .NET 7 Download page [Електронний ресурс] // Microsoft. – 2022. – Режим доступу до ресурсу: https://dotnet.microsoft.com/en-us/download/dotnet/7.0;

ДОДАТКИ

Додаток А

Програмна реалізація протоколу

```
using FluentAssertions;
using QuantoCrypt.Connection;
using QuantoCrypt.Infrastructure.CipherSuite;
using QuantoCrypt.Infrastructure.Connection;
using QuantoCrypt.Internal.CipherSuite;
using QuantoCrypt.Internal.Connection;
using System.Net;
using System.Net.Sockets;
using System.Text;

namespace QuantoCrypt
{
    class Program
    {
        static void Main(string[] args)
        {
            //Console.WriteLine("test");

            // Get Host IP Address that is used to establish a
connection
            // In this case, we get one IP address of localhost
that is IP : 127.0.0.1
            // If a host has multiple addresses, you will get a
list of addresses
            IPHostEntry host = Dns.GetHostEntry("localhost");
            IPAddress ipAddress = host.AddressList[0];
            IPEndPoint localEndPoint = new IPEndPoint(ipAddress,
11000);


_sUseGeneralConnectionModeDemo(ipAddress.AddressFamily,
localEndPoint);

            localEndPoint.Port += 1;
            _sUseFastConnectionModeDemo(ipAddress.AddressFamily,
localEndPoint);

            localEndPoint.Port += 1;

_sUseFastShortConnectionModeDemo(ipAddress.AddressFamily,
localEndPoint);

            localEndPoint.Port += 1;

_sUseFastShortConnectionModeWithFallbackDemo(ipAddress.AddressFami
ly, localEndPoint);

            Console.WriteLine("Demo finished.");
```

```
            Console.ReadLine();
        }

        private static void
_sUseGeneralConnectionModeDemo(AddressFamily addressFamily,
EndPoint targetEndPoint)
        {
            Socket server = new Socket(addressFamily,
SocketType.Stream, ProtocolType.Tcp);
            // A Socket must be associated with an endpoint using
the Bind method
            server.Bind(targetEndPoint);
            // Specify how many requests a Socket can listen
before it gives Server busy response.
            // We will listen 10 requests at a time
            server.Listen(10);

            Socket client = new Socket(addressFamily,
SocketType.Stream, ProtocolType.Tcp);
            client.Connect(targetEndPoint);

            Action<string> traceAction = Console.WriteLine;

            // get SocketTransportConnection for client and
server.
            SocketTransportConnection serverCon = new
SocketTransportConnection(server.Accept(), traceAction, true);
            SocketTransportConnection clientCon = new
SocketTransportConnection(client, traceAction, true);

            // init QuantoCryptConnectionFactory, create
ISecureTransportConnection for client + server.
            ICipherSuiteProvider cipherSuiteProvider = new
QuantoCryptCipherSuiteProvider();

            QuantoCryptConnectionFactory factory = new
QuantoCryptConnectionFactory(cipherSuiteProvider);

            var serverStartTask = Task.Run(() =>
factory.CreateSecureServerConnection(serverCon));

            ISecureTransportConnection secureClient =
factory.CreateSecureClientConnection(clientCon, new
CrystalsKyber1024Aes_CrystalsDilithium5Aes_AesGcm());

            ISecureTransportConnection secureServer =
serverStartTask.Result;

            // send and recieve data.
            string testData = "test message";

            secureClient.Send(Encoding.UTF8.GetBytes(testData));
```

```
            var recievedMessage =
Encoding.UTF8.GetString(secureServer.Receive());

            secureClient.Close();
            secureServer.Close();

            Action act1 = () =>
secureClient.Send(Encoding.UTF8.GetBytes(testData));
            Action act2 = () => secureServer.Receive();

            act1.Should().Throw<ObjectDisposedException>()
                .WithMessage("*The connection has been
successfully closed and can't be used!*");
            act2.Should().Throw<ObjectDisposedException>()
                .WithMessage("*The connection has been
successfully closed and can't be used!*");

            recievedMessage.Should().Be(testData);
        }

        private static void
_sUseFastConnectionModeDemo(AddressFamily addressFamily, EndPoint
targetEndPoint)
        {
            Socket server = new Socket(addressFamily,
SocketType.Stream, ProtocolType.Tcp);
            // A Socket must be associated with an endpoint using
the Bind method
            server.Bind(targetEndPoint);
            // Specify how many requests a Socket can listen
before it gives Server busy response.
            // We will listen 10 requests at a time
            server.Listen(10);

            Socket client = new Socket(addressFamily,
SocketType.Stream, ProtocolType.Tcp);
            client.Connect(targetEndPoint);

            Action<string> traceAction = Console.WriteLine;

            // get SocketTransportConnection for client and
server.
            SocketTransportConnection serverCon = new
SocketTransportConnection(server.Accept(), traceAction, true);
            SocketTransportConnection clientCon = new
SocketTransportConnection(client, traceAction, true);

            // init QuantoCryptConnectionFactory, create
ISecureTransportConnection for client + server.
            ICipherSuiteProvider cipherSuiteProvider = new
QuantoCryptCipherSuiteProvider();
```

```
            QuantoCryptConnectionFactory factory = new
QuantoCryptConnectionFactory(cipherSuiteProvider);

            var serverStartTask = Task.Run(() =>
factory.CreateSecureServerConnection(serverCon));

            ISecureTransportConnection secureClient =
factory.CreateSecureClientConnection(clientCon, new
CrystalsKyber1024Aes_CrystalsDilithium5Aes_Aes(),
QuantoCryptConnection.ConnectionMode.Fast);

            ISecureTransportConnection secureServer =
serverStartTask.Result;

            // send and recieve data.
            string testData = "test message";

            secureClient.Send(Encoding.UTF8.GetBytes(testData));

            var recievedMessage =
Encoding.UTF8.GetString(secureServer.Receive());

            secureClient.Close();
            secureServer.Close();

            Action act1 = () =>
secureClient.Send(Encoding.UTF8.GetBytes(testData));
            Action act2 = () => secureServer.Receive();

            act1.Should().Throw<ObjectDisposedException>()
                .WithMessage("*The connection has been
successfully closed and can't be used!*");
            act2.Should().Throw<ObjectDisposedException>()
                .WithMessage("*The connection has been
successfully closed and can't be used!*");

            recievedMessage.Should().Be(testData);
        }

        private static void
_sUseFastShortConnectionModeDemo(AddressFamily addressFamily,
EndPoint targetEndPoint)
        {
            Socket server = new Socket(addressFamily,
SocketType.Stream, ProtocolType.Tcp);
            // A Socket must be associated with an endpoint using
the Bind method
            server.Bind(targetEndPoint);
            // Specify how many requests a Socket can listen
before it gives Server busy response.
            // We will listen 10 requests at a time
            server.Listen(10);
```

```
            Socket client = new Socket(addressFamily,
SocketType.Stream, ProtocolType.Tcp);
            client.Connect(targetEndPoint);

            Action<string> traceAction = Console.WriteLine;

            // get SocketTransportConnection for client and
server.
            SocketTransportConnection serverCon = new
SocketTransportConnection(server.Accept(), traceAction, true);
            SocketTransportConnection clientCon = new
SocketTransportConnection(client, traceAction, true);

            // init QuantoCryptConnectionFactory, create
ISecureTransportConnection for client + server.
            ICipherSuiteProvider cipherSuiteProvider = new
QuantoCryptCipherSuiteProvider();

            QuantoCryptConnectionFactory factory = new
QuantoCryptConnectionFactory(cipherSuiteProvider);

            var serverStartTask = Task.Run(() =>
factory.CreateSecureServerConnection(serverCon));

            ISecureTransportConnection secureClient =
factory.CreateSecureClientConnection(clientCon, new
CrystalsKyber1024Aes_CrystalsDilithium5Aes_AesGcm(),
QuantoCryptConnection.ConnectionMode.FastShort);

            ISecureTransportConnection secureServer =
serverStartTask.Result;

            // send and recieve data.
            string testData = "test message";

            secureClient.Send(Encoding.UTF8.GetBytes(testData));

            var recievedMessage =
Encoding.UTF8.GetString(secureServer.Receive());

            secureClient.Close();
            secureServer.Close();

            Action act1 = () =>
secureClient.Send(Encoding.UTF8.GetBytes(testData));
            Action act2 = () => secureServer.Receive();

            act1.Should().Throw<ObjectDisposedException>()
                .WithMessage("*The connection has been
successfully closed and can't be used!*");
            act2.Should().Throw<ObjectDisposedException>()
                .WithMessage("*The connection has been
successfully closed and can't be used!*");
```

```
            recievedMessage.Should().Be(testData);
        }

        private static void
_sUseFastShortConnectionModeWithFallbackDemo(AddressFamily
addressFamily, EndPoint targetEndPoint)
        {
            Socket server = new Socket(addressFamily,
SocketType.Stream, ProtocolType.Tcp);
            // A Socket must be associated with an endpoint using
the Bind method
            server.Bind(targetEndPoint);
            // Specify how many requests a Socket can listen
before it gives Server busy response.
            // We will listen 10 requests at a time
            server.Listen(10);

            Socket client = new Socket(addressFamily,
SocketType.Stream, ProtocolType.Tcp);
            client.Connect(targetEndPoint);

            Action<string> traceAction = Console.WriteLine;

            // get SocketTransportConnection for client and
server.
            SocketTransportConnection serverCon = new
SocketTransportConnection(server.Accept(), traceAction, true);
            SocketTransportConnection clientCon = new
SocketTransportConnection(client, traceAction, true);

            // init QuantoCryptConnectionFactory, create
ISecureTransportConnection for client + server.
            List<ICipherSuite> serverSupportedCipherSuites = new
List<ICipherSuite>()
            {
                new CrystalsKyber1024_CrystalsDilithium5_Aes(),
                new CrystalsKyber1024_CrystalsDilithium5_AesGcm(),
                new CrystalsKyber1024_CrystalsDilithium5Aes_Aes(),
                new
CrystalsKyber1024_CrystalsDilithium5Aes_AesGcm()
            };
            List<ICipherSuite> clientSupportedCipherSuites = new
List<ICipherSuite>()
            {
                new CrystalsKyber1024_CrystalsDilithium5_Aes(),
                new CrystalsKyber1024_CrystalsDilithium5_AesGcm(),
                new CrystalsKyber1024_CrystalsDilithium5Aes_Aes(),
                new
CrystalsKyber1024_CrystalsDilithium5Aes_AesGcm(),
                new CrystalsKyber1024Aes_CrystalsDilithium5_Aes(),
                new
CrystalsKyber1024Aes_CrystalsDilithium5_AesGcm()
```

```
            };
            ICipherSuiteProvider serverCipherSuiteProvider = new
CustomCipherSuiteProvider(serverSupportedCipherSuites);
            ICipherSuiteProvider clientCipherSuiteProvider = new
CustomCipherSuiteProvider(clientSupportedCipherSuites);

            var serverStartTask = Task.Run(() =>
QuantoCryptConnection.InitializeSecureServer(serverCipherSuiteProv
ider, serverCon));

            ISecureTransportConnection secureClient =
QuantoCryptConnection.InitializeSecureClient(clientCipherSuiteProv
ider, new CrystalsKyber1024Aes_CrystalsDilithium5_AesGcm(),
clientCon, QuantoCryptConnection.ConnectionMode.FastShort);

            ISecureTransportConnection secureServer =
serverStartTask.Result;

            // send and recieve data.
            string testData = "test message";

            secureClient.Send(Encoding.UTF8.GetBytes(testData));

            var recievedMessage =
Encoding.UTF8.GetString(secureServer.Receive());

            secureClient.Close();
            secureServer.Close();

            Action act1 = () =>
secureClient.Send(Encoding.UTF8.GetBytes(testData));
            Action act2 = () => secureServer.Receive();

            act1.Should().Throw<ObjectDisposedException>()
                .WithMessage("*The connection has been
successfully closed and can't be used!*");
            act2.Should().Throw<ObjectDisposedException>()
                .WithMessage("*The connection has been
successfully closed and can't be used!*");

            recievedMessage.Should().Be(testData);
        }

        private void
_sTestViaFluentAPIWithMultipleInstances(AddressFamily
addressFamily, EndPoint targetEndPoint)
        {
            Action<string> traceAction = Console.WriteLine;
            SocketTransportConnection serverCon =
SocketTransportConnection.CreateDefaultServer(addressFamily,
targetEndPoint, traceAction);
```

```
            SocketTransportConnection clientCon1 =
SocketTransportConnection.CreateDefaultClient(addressFamily,
targetEndPoint, traceAction);

            var activeServerConnection1 = serverCon.Connect();

            ICipherSuiteProvider cipherSuiteProvider = new
QuantoCryptCipherSuiteProvider();

            QuantoCryptConnectionFactory factory = new
QuantoCryptConnectionFactory(cipherSuiteProvider);

            var serverStartTask = Task.Run(() =>
factory.CreateSecureServerConnection(activeServerConnection1));

            ISecureTransportConnection secureClient1 =
factory.CreateSecureClientConnection(clientCon1);
            ISecureTransportConnection secureServer1 =
serverStartTask.Result;

            // send and recieve data.
            string testData = "test message";

            // send from client1 to server1.
            secureClient1.Send(Encoding.UTF8.GetBytes(testData));
            var recievedMessage =
Encoding.UTF8.GetString(secureServer1.Receive());

            if (!recievedMessage.Substring(0,
testData.Length).Equals(testData,
StringComparison.OrdinalIgnoreCase))
                throw new Exception($"Recieved massage
[{recievedMessage}] != expected [{testData}].");

            // create 2nd connection pair.
            SocketTransportConnection clientCon2 =
SocketTransportConnection.CreateDefaultClient(addressFamily,
targetEndPoint, traceAction);
            var activeServerConnection2 = serverCon.Connect();

            serverStartTask = Task.Run(() =>
factory.CreateSecureServerConnection(activeServerConnection2));

            ISecureTransportConnection secureClient2 =
factory.CreateSecureClientConnection(clientCon2);
            ISecureTransportConnection secureServer2 =
serverStartTask.Result;

            // send from client2 to server2.
            secureClient2.Send(Encoding.UTF8.GetBytes(testData));
            var recievedMessage2 =
Encoding.UTF8.GetString(secureServer2.Receive());
```

```
            if (!recievedMessage2.Substring(0,
testData.Length).Equals(testData,
StringComparison.OrdinalIgnoreCase))
                throw new Exception($"Recieved massage
[{recievedMessage2}] != expected [{testData}].");

            // send from client1 to server1.
            secureClient1.Send(Encoding.UTF8.GetBytes(testData));
            var recievedMessage3 =
Encoding.UTF8.GetString(secureServer1.Receive());

            recievedMessage.Should().Be(testData);
        }
    }
}
```