Дослідження методів захищеної передачі даних у локальній мережі

Research of Methods of Secured Data Transmission in a Local Network

Михайло Соколовський [A]
студент кафедри захисту інформації, e-mail: mykhailo.sokolovskyi.mkbst.2024@lpnu.ua, ORCID ID: 0009-0007-3726-2273

Максим Турега [A]
Corresponding author: студент кафедри захисту інформації, e-mail: maksym.tureha.mkbst.2024@lpnu.ua, ORCID ID: 0009-0006-5121-166X

Марія Швед [A]
кандидат технічних наук, старший викладач кафедри захисту інформації, e-mail: mariia.y.shved@lpnu.ua, ORCID ID: 0000-0003-0428-7777

Mykhailo Sokolovskyi [A]
Student of the Department of Information Security, e-mail: mykhailo.sokolovskyi.mkbst.2024@lpnu.ua, ORCID ID: 0009-0007-3726-2273

Maksym Tureha [A]
Corresponding author: Student of the Department of Information Security, e-mail: maksym.tureha.mkbst.2024@lpnu.ua, ORCID ID: 0009-0006-5121-166X

Mariia Shved [A]
Candidate of Technical Sciences, Senior Lecturer of Department of Information Security, e-mail: mariia.y.shved@lpnu.ua, ORCID ID: 0000-0003-0428-7777

[A] Національний університет "Львівська політехніка", м. Львів, Україна

[A] Lviv Polytechnic National University, Lviv, Ukraine

Received: October 20, 2025 | Revised: October 30, 2025 | Accepted: October 31, 2025

DOI: https://doi.org/10.33445/sds.2025.15.5.17

Мета роботи. Дослідження методів і принципів забезпечення захищеної передачі даних у локальній мережі та розроблення програмної утиліти для встановлення захищеного каналу зв'язку.

Метод дослідження. Ґрунтується на використанні криптографічного алгоритму AES та протоколу обміну ключами Діффі-Хеллмана, а також на застосуванні перетворень байтових послідовностей під час генерації спільного секретного ключа.

Результати дослідження. Аналіз принципів функціонування наскрізного шифрування (End-to-End Encryption, E2EE), розгляді криптографічних алгоритмів, що застосовуються в межах E2EE, та дослідженні загроз типу «людина посередині» (Man-in-the-Middle, MITM) для таких систем. У межах практичної частини реалізовано утиліту, що забезпечує захищену передачу даних у локальній мережі.

Теоретична цінність дослідження. Поглиблення розуміння механізмів криптографічного захисту при організації безпечного обміну даними в локальних мережах.

Практична цінність дослідження. Створення прикладної програмної реалізації, яка може бути використана для формування захищеного каналу зв'язку в умовах загальнодоступних або потенційно вразливих локальних мереж.

Цінність дослідження. Можливість використання запропонованого підходу під час проектування або модернізації систем захищеного обміну даними в мережевій інфраструктурі різного масштабу.

Тип статті. Прикладна.

Purpose. Study of methods and principles of ensuring secure data transmission in a local network and development of a software utility for establishing a secure communication channel.

Method. Based on the use of the AES cryptographic algorithm and the Diffie-Hellman key exchange protocol, as well as the application of byte sequence transformations during the generation of a shared secret key.

Findings. Consist in the analysis of the principles of functioning of end-to-end encryption (E2EE), consideration of cryptographic algorithms used within E2EE, and the study of Man-in-the-Middle (MITM) threats for such systems. Within the practical part, a utility was implemented that ensures secure data transmission in a local network.

Theoretical implications. Consists in deepening the understanding of cryptographic protection mechanisms when organizing secure data exchange in local networks.

Practical implications. Consists in creating an applied software implementation that can be used to form a secure communication channel in the conditions of publicly accessible or potentially vulnerable local networks.

The value. Consists in the possibility of using the proposed approach when designing or upgrading secure data exchange systems in network infrastructure of various scales.

Paper type. Applied.

Ключові слова: шифрування, захищена передача даних, криптографічні протоколи, захищений канал зв'язку, E2EE.

Key words: encryption, secure data transmission, cryptographic protocols, secure communication channel, E2EE.

Вступ

Захищена передача даних є важливою складовою кібербезпеки. В історії людства спостерігається постійні спроби максимально приховати інформацію, що передається: від шифру Цезаря до електромеханічної шифрувальної машини "Enigma", яка широко застосовувалась під час Другої світової війни [1]. Усі ці методи об'єднувала одна загальна ціль — безпечно та конфіденційно передати таємну інформацію відкритими каналами зв'язку. В

основі своїй для кожного з них потрібна була певна інформація — ключ, за допомогою якого можливе шифрування та дешифрування повідомлення. Зазвичай основною проблемою була безпечна передача цього ключа, що і визначає, чи дійсно канал зв'язку є захищеним. Тому саме на ключ шифрування часто спрямовувались зусилля тих, хто перехоплював інформацію.

З появою електронно-обчислювальних машин (ЕОМ) класичні методи шифрування відходять у минуле, хоча їхні цілі й принципи залишаються актуальними. ЕОМ дозволили обробляти інформацію в масштабах, які раніше були недоступні. З появою комп'ютерів стало можливим шифрувати будь-які дані, представлені у двійковому форматі, на відміну від класичних шифрів, призначених переважно для текстових повідомлень. Це значно знизило ефективність лінгвістичних методів криптоаналізу. Багато сучасних шифрів працюють з послідовностями бітів (часто — у вигляді блоків), тоді як класичні або механічні системи переважно оперували літерами. Проте комп'ютери також отримали застосування у криптоаналізі, що в певній мірі компенсує зростання складності самих алгоритмів. У відповідь на стрімкий розвиток обчислювальних потужностей алгоритми шифрування стали складнішими й обчислювально витратнішими, проте це дозволило захищати значно більші обсяги інформації, ніж будь-коли до того [2]. Зі зростанням ролі ЕОМ також розвивалися комп'ютерні мережі для їхньої взаємодії, хоча на початкових етапах дані передавалися у відкритому форматі, оскільки контроль над доступом до мереж був достатньо жорстким, і більша увага приділялась фізичному захисту інформації на кінцевих точках. Але із поширенням портативних комп'ютерів та глобальної мережі Internet з'явилася необхідність застосовувати шифрування для передачі даних у загальнодоступних мережах, щоб гарантувати конфіденційність інформації. У першу чергу потреба у цьому виникла в оборонній сфері, оскільки мережами передавалась значна кількість секретної інформації.

У сучасному світі комп'ютерні мережі охоплюють практично всі сфери людської діяльності. Через них передаються значні обсяги важливої, конфіденційної або критично важливої для безпеки інформації — від даних транзакцій до медичних записів, від конфіденційних документів до інформації з грифом, що має значення для національної безпеки. Цей факт створює постійний стимул для розвитку методів шифрування, обміну ключами та їхнього впровадження в різних типах комп'ютерних мереж і альтернативних каналах зв'язку. Захист таких мереж буде залишатись актуальним незалежно від епохи. Проте, якщо ви читали книги з комп'ютерної безпеки, то, мабуть, зустрічалися з поширеним поглядом на криптографію. "Криптографія, – кажуть автори, – найміцніша ланка в ланцюжку". Такий погляд на криптографію ідеалізований, та насправді є міфом. Криптографія міцна в теорії, але на практиці так само схильна до помилок, як і будь-який інший аспект системи безпеки (зокрема через людський фактор, неправильну конфігурацію чи реалізацію) [3].

Теоретичні основи дослідження

End-to-end encryption (E2EE), або наскрізне шифрування, – це технологія захисту даних, що виникла в процесі розвитку криптографії та методів безпечної передачі інформації. Суть E2EE полягає в тому, що доступ до ключів шифрування та повідомлень мають лише користувачі, які беруть участь у спілкуванні. Використання цього методу шифрування унеможливлює доступ до ключів або змісту повідомлень третім особам, включно з провайдерами сервісів, що забезпечує високий рівень конфіденційності [4]. На сьогодні наскрізне шифрування є одним із найбільш надійних і поширених способів захисту інформації в цифрових комунікаціях, зокрема в месенджерах, електронній пошті та хмарних сервісах.

Загальний принцип роботи наскрізного шифрування полягає в тому, що до вихідних даних, будь-то повідомлення чи файли, мають доступ лише відправник і одержувач. Інформація стає конфіденційною та недоступною для провайдерів, серверів і сторонніх осіб, через які здійснюється передача даних у загальнодоступній мережі. Для реалізації E2EE на

кінцевих пристроях необхідно попередньо обмінятися ключами шифрування. Після цього дані шифруються на пристрої відправника, передаються мережею в зашифрованому вигляді та розшифровуються лише на пристрої одержувача, відтворюючи їх у вихідному форматі. Таким чином, протягом усього маршруту передавання дані залишаються зашифрованими й недоступними для сторонніх осіб, забезпечуючи високий рівень конфіденційності та захисту.

Ідея та розвиток наскрізного шифрування почали формуватися зі створенням асинхронного обміну повідомленнями через SMTP (Simple Mail Transfer Protocol) [5], оскільки на той час не існувало ефективних методів, які б забезпечували конфіденційність електронної пошти. У 1991 році було розроблено пакет програмного забезпечення PGP (Pretty Good Privacy) [6], що реалізував наскрізне шифрування та електронний підпис для електронних листів. PGP швидко здобув популярність серед користувачів, але його впровадження як стандарту E2EE для електронної пошти відбулося лише в 1997 році під назвою OpenPGP [7]. На сьогодні OpenPGP застосовується у різних поштових клієнтах і мобільних додатках, таких як Enigmail для Thunderbird, iPGMail для iOS, а також у системі управління ключами OpenKeychain для Android. Технологія E2EE стала логічним продовженням і розвитком концепції, закладеної у PGP, розширюючи можливості захисту даних у цифрових комунікаціях.

В основі сучасних месенджерів широко використовується протокол OTR (Off-the-Record Messaging), який був розроблений у 2004 році. OTR є розширенням XMPP – відкритого мережевого протоколу для миттєвого обміну повідомленнями та інформацією про користувачів у мережі. Цей криптографічний протокол забезпечує E2EE у програмах для швидкого обміну повідомленнями. Протокол OTR поєднує AES-шифрування, протокол Діффі-Геллмана та хеш-функцію SHA-1, що гарантує конфіденційність та цілісність переданих даних. Однією з головних переваг протоколу є неможливість ідентифікації користувачів за ключами та доступу до вихідних повідомлень [8, 9]. OTR застосовується у сучасних месенджерах, таких як Telegram, Viber та WhatsApp. Ще одним сучасним протоколом є Signal, розроблений компанією Open Whisper Systems для забезпечення наскрізного шифрування голосових викликів, відеодзвінків, миттєвих повідомлень та інших даних. Протокол використовує Double Ratchet Algorithm і розширений протокол потрійного обміну ключами Діффі-Хеллмана (3-DH), а також криптографічні примітиви Curve25519, AES-256 і HMAC-SHA256. Протокол застосовується в однойменному месенджері Signal [10] і забезпечує високий рівень безпеки.

Огляд алгоритмів шифрування, що застосовуються у E2EE. Основними алгоритмами шифрування у E2EE, є симетричний алгоритм AES (Advanced Encryption Standard) та асиметричний алгоритм RSA. AES – це симетричний блочний алгоритм шифрування, який переміг у конкурсі та був прийнятий як стандарт шифрування урядом США у 2002 році. Станом на 2009 рік AES є одним із найпоширеніших алгоритмів симетричного шифрування [11].

AES використовує блочне шифрування, де вхідні дані розбиваються на блоки фіксованого розміру (128 біт у стандартній версії AES). Кожен блок обробляється окремо з використанням ключа шифрування, а результат стає вхідним для наступного блоку. Алгоритм оперує станом, який представляє собою масив 4×4 байт. Версії AES із більшими блоками включають додаткові колонки в масиві стану. Процес шифрування відбувається у раундах, кожен з яких є основною одиницею обробки даних. У кожному раунді виконуються чотири послідовні операції:

1. SubBytes – заміна кожного байта вхідного блоку на відповідний байт із таблиці замін, яка формується на основі ключа шифрування.
2. ShiftRows – перестановка байтів у кожному рядку вхідного блоку на фіксований зсув, залежно від номера рядка.
3. MixColumns – лінійне перетворення кожного стовпця вхідного блоку за допомогою фіксованої матриці, що забезпечує дифузію даних.

4. AddRoundKey – комбінування вхідного блоку з частинами ключа шифрування шляхом операції додавання.

У останньому раунді операція MixColumns не виконується, а дані стають остаточно зашифрованими. Така структура забезпечує стійкість до сучасних атак криптоаналізу.

Основні переваги AES перед іншими алгоритмами шифрування включають високу ефективність, безпеку та надійність. Алгоритм підтримує різні довжини ключів – 128, 192 та 256 біт, що забезпечує стійкість до атак методом перебору ключів (brute force). Довжина ключа безпосередньо впливає на кількість раундів шифрування: 128 біт – 10 раундів, 192 біт – 12 раундів, 256 біт – 14 раундів, що дозволяє адаптувати рівень безпеки залежно від вимог до захисту даних [3, 12]. Так як, AES є офіційним стандартом національної безпеки США (National Institute of Standards and Technology, NIST), це забезпечує його застосування в урядових установах та організаціях з підвищеними вимогами до безпеки. Завдяки комбінації високої продуктивності, гнучкості у виборі ключів та перевіреної надійності AES є найбільш поширеним алгоритмом симетричного шифрування у світі, включно з використанням у системах E2EE.

RSA (від прізвищ його розробників Rivest, Shamir та Adleman) – це алгоритм шифрування з відкритим ключем, що базується на факторизації великих цілих чисел. RSA став першим практично застосовним асиметричним алгоритмом, придатним як для шифрування, так і для створення електронного підпису [13]. Принцип роботи алгоритму RSA можна описати наступними кроками: (1) згенерувати два простих числа p та q; (2) Обчислити їхній добуток $n = p \cdot q$; (3) Обчислити функцію Ейлера $\varphi(n) = (p-1)(q-1)$; (4) Вибрати секретну експоненту d, яка задовольняє умову $d \cdot e \equiv 1 \pmod{\varphi(n)}$, де e – відкрита експонента; (5) Опублікувати відкритий ключ (e, n) та зберегти приватний ключ (d, n) у таємниці; (6) Вибрати дані для шифрування та обчислити шифротекст за допомогою відкритого ключа: $C = M^e \bmod n$, де M – повідомлення; (7) Розшифрувати повідомлення за допомогою приватного ключа: $M = C^d \bmod n$.

Варто зауважити, що RSA у порівнянні з AES потребує значно більших обчислювальних ресурсів та часу для шифрування, тому для швидкого обміну повідомленнями доцільніше застосовувати симетричні алгоритми (наприклад, AES), а для передавання важливих повідомлень або конфіденційних даних – асиметричні алгоритми [13].

Методи та алгоритми обміну ключами шифрування у мережі. Наразі можна виокремити два основні методи обміну ключами шифрування: за допомогою алгоритму RSA та протоколу Діффі-Хеллмана. Найпростішим методом є обмін публічними ключами в алгоритмі RSA. Під час такого обміну відправнику надсилається публічний ключ тієї сторони, яка буде шифрувати дані. Передача ключа може здійснюватися різними способами: електронною поштою, через месенджер або навіть на електронному носії при особистій передачі. Для цього необхідно мати певну попередню інформацію про того, з ким планується обмін даними. Модель процесу обміну ключами в RSA наведено на рис. 1.

З точки зору простоти, цей метод є найбільш доступним, оскільки не вимагає складних алгоритмічних обчислень. Проте існують потенційні вразливості, що можуть поставити під загрозу безпечність передачі даних. Такі загрози включають можливість підробки або перехоплення публічного ключа зловмисником, що може призвести до компрометації каналу зв’язку. Детальніше ці загрози будуть розглянуті далі [14].

Протокол Діффі-Хеллмана – це метод узгодження ключів шифрування у мережі, учасники якої не мають попередньої інформації один про одного. На відміну від RSA, де сторони обміну повинні знати певні відомості про ключі та учасників, протокол Діффі-Хеллмана дозволяє здійснити узгодження ключів повністю анонімно. Для отримання спільного ключа між кінцевими точками протокол вимагає виконання певних алгоритмічних обчислень, що забезпечує безпечну генерацію секретного ключа навіть у відкритій мережі. Зазвичай цей метод використовується для обміну ключами симетричних алгоритмів

шифрування, таких як AES, що дозволяє згодом ефективно шифрувати дані. Протокол Діффі-Хеллмана забезпечує високий рівень конфіденційності, оскільки ключ ніколи не передається мережею у відкритому вигляді. Проте його ефективність залежить від розміру чисел, що використовуються, та обчислювальних потужностей кінцевих пристроїв [15].

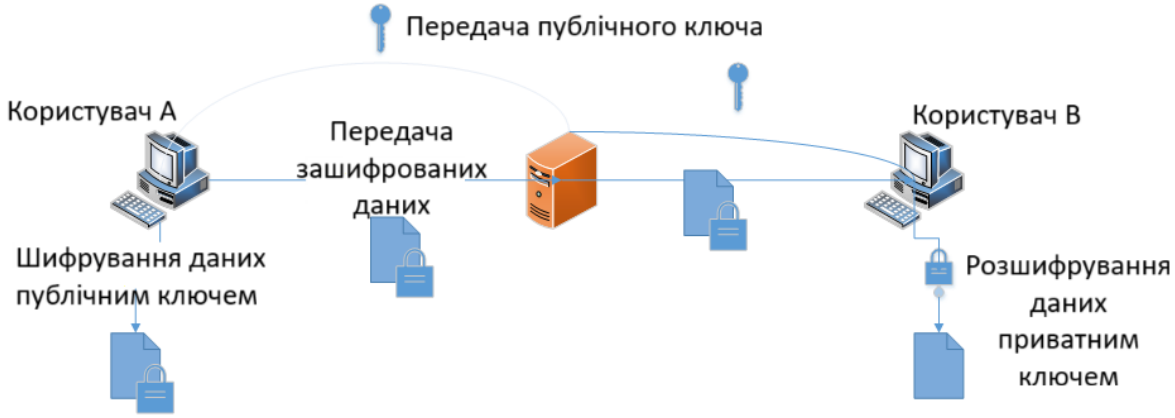

Рисунок 1 – Візуальне представлення безпечної передачі даних за допомогою RSA

Атака MITM в E2EE. Атака “людина посередині” (MITM) — це вид мережевої атаки, при якій зловмисник перехоплює та/або перенаправляє трафік між двома сторонами, що спілкуються, при цьому для жертв ця проміжна присутність залишається непоміченою. Такі атаки дозволяють нападнику перехоплювати, підмінювати або підслуховувати повідомлення, що серйозно підриває конфіденційність і цілісність обміну даними.

У контексті алгоритмів шифрування та обміну ключами MITM становить особливу загрозу: під час передачі або обміну публічними ключами зловмисник може перехопити повідомлення і замінити публічний ключ однієї зі сторін на свій власний (key substitution). У цьому випадку відправник помилково шифрує дані ключем нападника, який, у свою чергу, може їх розшифрувати, прочитати й за необхідності повторно зашифрувати для одержувача — таким чином зловмисник отримує доступ до всього шифрованого трафіку (рис. 2.). Цей механізм використовують для компрометації сесій, викрадення облікових даних або доступу до банківських кабінетів та іншої конфіденційної інформації [16].

Технічно MITM-атаку можна реалізувати різними способами. Для локальних мереж типовою технікою є ARP-спуфінг (ARP poisoning / ARP spoofing), при якому атакуючий підмінює ARP-записи в кешах хостів, змушуючи їх відправляти трафік через вузол нападника. Інші методи включають підробку DNS-відповідей, створення фіктивних (rogue) Wi-Fi-точок доступу, SSL/TLS-стріппінг або примусове встановлення “довірених” кореневих сертифікатів на клієнтських пристроях [17, 18]. Як наслідок, захист від MITM вимагає як криптографічних заходів (наприклад, автентифікація ключів через довірені PKI, перевірені fingerprint-значення, використання протоколів з автентифікацією на етапі обміну ключами). MITM атаки можна використати для злому ключів шифрування, наприклад при визначені прогалин в генераторах псевдовипадкових чисел, що використовуються при створенні ключів RSA. Виявлені вади роблять практично досяжною факторизацію використаних при генерації ключів простих чисел. Зловмисник має можливість відновити секретний ключ за відкритим ключем жертви [14].

Методи та засоби захисту від атак на E2EE. Електронний цифровий підпис (ЕЦП) є першочерговим засобом для захисту та безпечної передачі ключів у мережі. Використання ЕЦП дозволяє переконатися, що публічний ключ дійсно надіслано саме тією стороною, з якою встановлено зв’язок, та запобігти підміні або підробці ключів під час обміну [19].

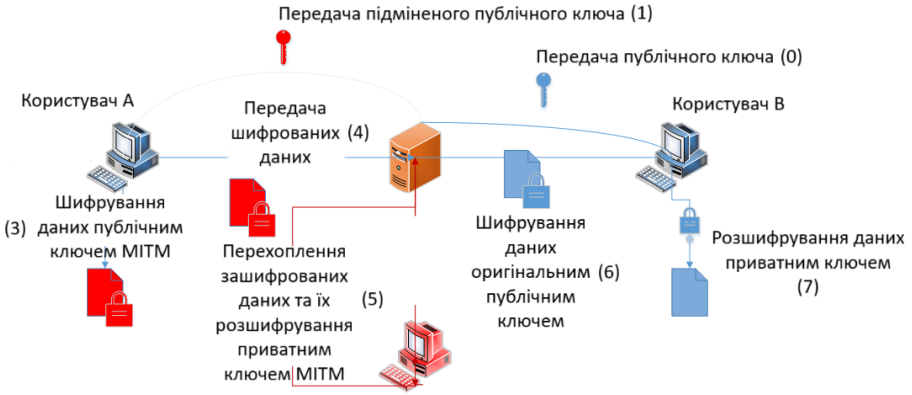

Рисунок 2 – Схема проведення MITM атаки на підміну ключів шифрування

Щоб уникнути вразливостей, пов’язаних із ненадійною роботою генераторів псевдовипадкових чисел, рекомендовано використовувати апаратні можливості процесора для генерування істинно випадкових чисел (апаратні генератори випадкових чисел). Це підвищує стійкість ключів до атак методом підбору або відтворення. Однак не всі платформи й процесори підтримують такі можливості, тому в рамках цього дослідження передбачено використання генератора псевдовипадкових чисел як уніфікованого рішення для сумісності з різними платформами.

Постановка проблеми

У сучасних локальних мережах обмін даними відбувається з високою інтенсивністю, що, у свою чергу, підвищує ризик несанкціонованого доступу, перехоплення трафіку та здійснення атак на передавані дані. Однією з найбільш поширених та небезпечних загроз є атака типу “людина посередині” (MITM), за якої зловмисник може перехопити та модифікувати інформацію під час її передачі. Попри наявність криптографічних механізмів захисту, існуючі рішення не завжди забезпечують належну безпеку в межах локальних мереж, та у випадках неправильного обміну ключами або відсутності перевірки автентичності сторін.

Таким чином, виникає потреба у створенні практичного інструменту, який би дозволяв підвищити рівень захищеності обміну даними в локальному середовищі. Основна проблема полягає в поєднанні ефективних криптографічних алгоритмів із зручними та надійними механізмами їх реалізації, зокрема із захистом від підміни ключів під час обміну. Це потребує комплексного підходу, що включає не лише шифрування даних, але й перевірку їх цілісності та автентичності, а також безпечне узгодження ключів між сторонами каналу зв’язку.

Методологія дослідження

Для генерації та обміну ключами шифрування в межах розроблюваної утиліти було обрано протокол Діффі–Хеллмана. Його використання дає змогу здійснювати узгодження спільного секретного ключа між двома або більше сторонами без необхідності попереднього обміну конфіденційною інформацією. Такий підхід забезпечує стійкість до перехоплення ключів, оскільки навіть у разі повного контролю трафіку зловмисник не зможе відтворити секретний ключ без знання приватних параметрів сторін. Для безпосереднього шифрування даних було обрано симетричний алгоритм AES, який характеризується високою швидкодією та криптографічною стійкістю. Завдяки використанню спільного секретного ключа, отриманого в результаті виконання протоколу Діффі–Хеллмана, AES може ефективно забезпечувати конфіденційність переданих повідомлень у режимі реального часу.

Загалом поєднання цих двох алгоритмів дозволяє реалізувати сучасну гібридну криптосистему, де асиметричний механізм використовується виключно для безпечного

узгодження ключів, а симетричний — для подальшого захищеного обміну даними. Це забезпечує як високий рівень захищеності, так і оптимальну продуктивність системи.

Результати

Для створення захищеного каналу зв’язку в системі використовується комбінований підхід, що базується на поєднанні алгоритму Діффі–Хеллмана для узгодження спільного секретного ключа та алгоритму AES для подальшого шифрування переданих даних. Процес встановлення захищеного з’єднання здійснюється покроково. Спочатку ініціатор з’єднання надсилає запит на встановлення захищеного каналу, передаючи службовий пакет із параметрами довжини ключа та своїми відкритими даними, необхідними для виконання протоколу обміну. Отримувач повинен підтвердити цей запит, після чого сторони переходять до генерування спільного секретного ключа шифрування за допомогою алгоритму Діффі–Хеллмана. Після успішного завершення узгодження ключів секретний ключ використовується як основа для подальшого симетричного шифрування даних за стандартом AES. Таким чином забезпечується висока стійкість з’єднання до підміни ключів та перехоплення трафіку, оскільки фактичні ключі шифрування ніколи не передаються мережею у відкритому вигляді. Схематичне представлення процесу ініціалізації захищеного з’єднання наведено на рис. 3.

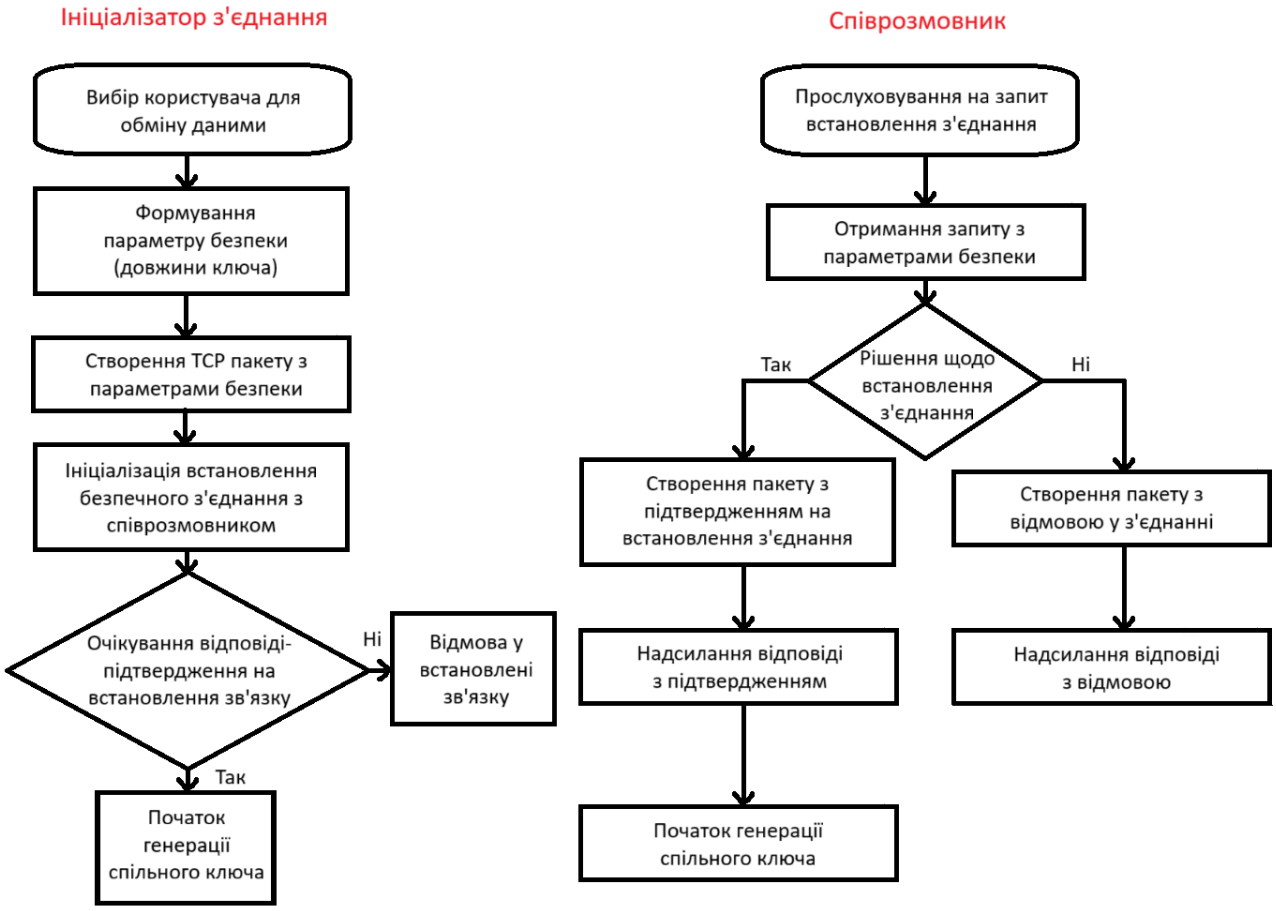

Рисунок 3 – Алгоритм ініціалізації встановлення захищеного з’єднання

Після завершення етапу ініціалізації розпочинається процес генерування спільного секретного ключа шифрування. Цей процес виконується окремою внутрішньою функцією системи та є повністю прозорим для користувача, тобто не потребує його участі й не відображається у вигляді жодних додаткових дій. Як зазначалося вище, узгодження ключів здійснюється за допомогою алгоритму Діффі–Хеллмана (DH). Для передавання службових

даних у цьому процесі використовується транспортний протокол TCP, що забезпечує гарантовану доставку пакетів та можливість підтвердження отримання (через механізм SYN/ACK). У відповіді, що надсилається ініціатору з'єднання, міститься другий відкритий параметр (Q), необхідний для обчислення спільного секретного ключа. На основі обміну цими параметрами обидві сторони обчислюють однаковий ключ шифрування, який у подальшому використовується алгоритмом AES для захисту трафіку. Схематичне зображення процесу узгодження та генерації ключа шифрування наведено на рис. 4.

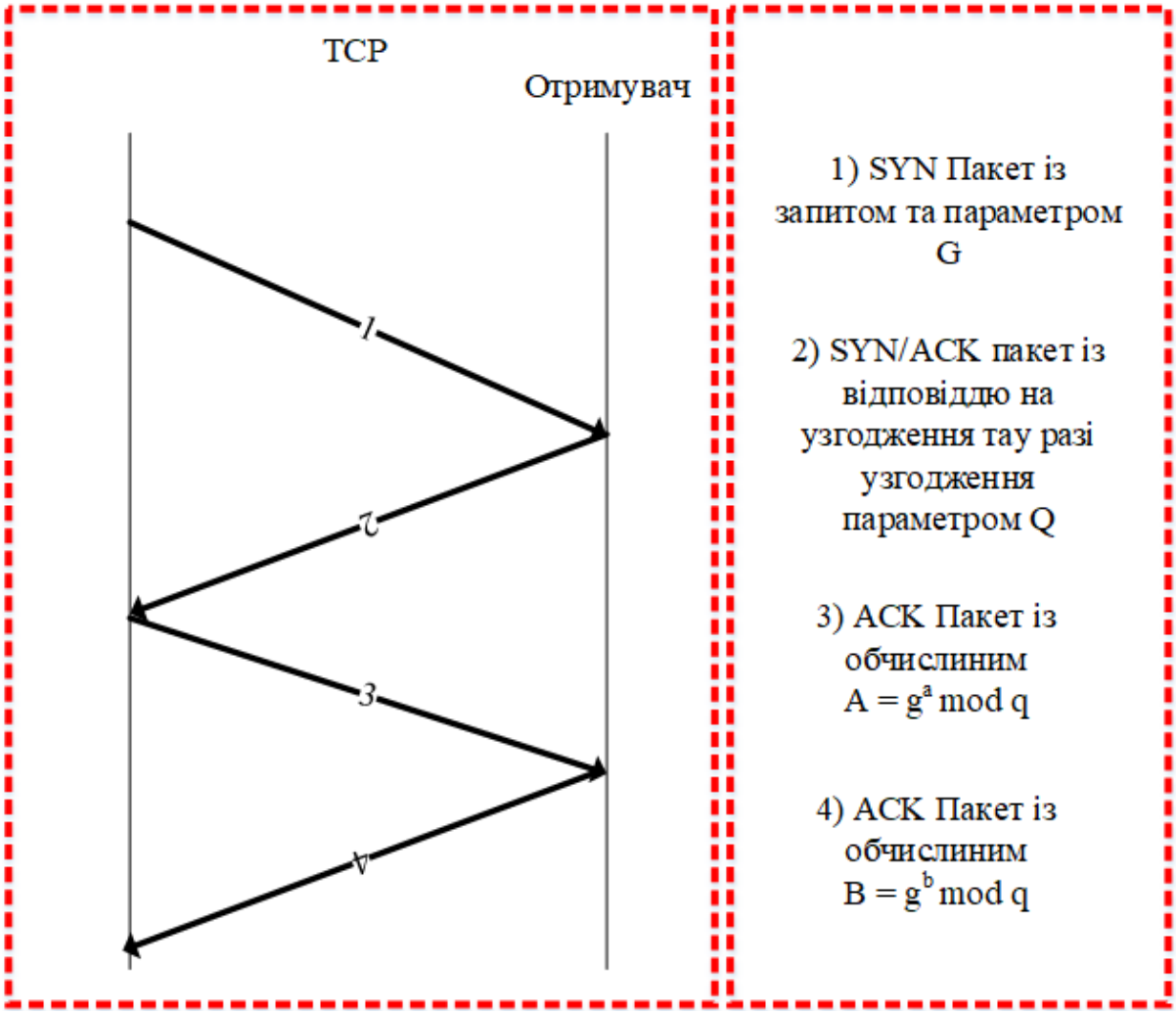

Рисунок 4 – Процес узгодження та генерації ключа в мережі за допомогою алгоритму DH

Після завершення процесу узгодження спільного секретного ключа встановлюється захищене з'єднання між сторонами. Отриманий ключ використовується для шифрування та дешифрування всіх даних, що передаються в межах цієї сесії. Водночас важливим є не лише факт його отримання, а й обмеження строку його дії, оскільки довготривале використання одного й того ж ключа знижує загальний рівень криптографічної стійкості. Для забезпечення додаткової безпеки в системі передбачене періодичне оновлення ключа шифрування: він автоматично регенерується кожні 30 хвилин або при кожному повторному встановленні з'єднання (тобто під час ініціації нової захищеної сесії). Такий підхід мінімізує ризик компрометації ключа та відповідає сучасним практикам криптографічного захисту інформації. Схематичний опис функціонування та підтримання захищеної сесії наведено на рис. 5.

Рисунок 5 – Процес підтримки або завершення захищеного обміну даними

Інструменти для розробки утиліти. Для створення утиліти було обрано мову програмування Python, оскільки вона є високорівневою, кросплатформеною та широко застосовується у сфері мережевої безпеки та криптографії. Python забезпечує швидку розробку програмних рішень, підтримує великий набір бібліотек і дозволяє легко комбінувати різні програмні модулі та функції. У межах проєкту використовувалася версія Python 3.x.x. Для реалізації основного функціоналу утиліти були задіяні такі бібліотеки:

- socket – використовується для створення та керування мережевими з’єднаннями;
- multiprocessing та threading – призначені для створення підпроцесів і потоків, що дозволяє виконувати паралельні операції та підвищувати продуктивність системи;
- pycryptodome – забезпечує реалізацію криптографічних алгоритмів, зокрема обмін ключами, шифрування та генерацію псевдовипадкових чисел.

Для розробки утиліти було використано середовище virtualenv, яке дозволяє створювати ізольоване середовище Python та уникати конфліктів між різними версіями бібліотек і пакетів. Це є важливим для стабільності розробки та повторюваності результатів. Розробка та тестування здійснювалися в операційних системах Kali Linux та Ubuntu, які є Unix-подібними дистрибутивами на базі Debian. Застосовуються в інфраструктурі кібербезпеки та підходять для роботи з мережевими інструментами та криптографічними додатками.

Огляд основного функціоналу утиліти. Процедура авторизації або реєстрації виконується щоразу при запуску програми та забезпечує ідентифікацію користувача в системі. Це дозволяє кожному користувачу мати власний обліковий запис і зберігати налаштування та

параметри доступу. Під час запуску утиліта пропонує вибір: зареєструватися або авторизуватися. У разі реєстрації введені користувачем ім’я (username) та пароль зберігаються у JSON-файлі користувачів. Паролі не зберігаються у відкритому вигляді — перед записом вони хешуються за допомогою алгоритму bcrypt, спеціально розробленого для захищеного зберігання паролів і стійкого до brute-force атак. Якщо ж такий користувач уже існує, система повідомляє про це й завершує виконання. Після успішної реєстрації користувач автоматично вважається авторизованим. Авторизація відбувається у два етапи: (1) перевірка наявності облікового запису; (2) порівняння хешу введеного паролю з хешем, що збережений у файлі. Якщо дані збігаються — доступ надано, і користувач може продовжити роботу з утилітою. У протилежному випадку програма завершує свою роботу.

Після успішної авторизації користувач обирає режим роботи: очікування отримання повідомлень або їх відправлення. На екран виводиться список усіх активних користувачів у локальній мережі, з якими можна встановити захищене з’єднання. Для цього запускається підпроцес, який періодично надсилає службові пакети про поточний стан користувача, тим самим сигналізуючи іншим вузлам про його активність. Серверна частина програми постійно прослуховує визначений порт і оновлює список активних користувачів: якщо новий вузол ще не зареєстрований у списку, він автоматично додається.

Після вибору співрозмовника ініціюється процедура узгодження криптографічних параметрів, і починається процес генерування спільного ключа шифрування за алгоритмом Діффі–Хеллмана, який буде використано для встановлення захищеного каналу зв’язку.

Генерація спільного ключа шифрування. Для формування спільного ключа шифрування використовується алгоритм Діффі–Хеллмана. Його реалізацію поділено на дві частини: клієнтську та серверну. Для генерації випадкових байтів застосовується функція get_random_bytes з бібліотеки Crypto.Random. Отримані байти перетворюються у числа типу integer, на основі яких обчислюється спільний секретний ключ. Довжину ключа задає ініціатор з’єднання — вона може становити 16, 24 або 32 байти. Формування кожного байту відбувається окремо, тому для кожного з них генерується нова узгоджена основа та секретні числа. Процес генерування спільного ключа включає такі кроки:

1. Клієнт надсилає серверу випадково згенероване число A;
2. Сервер отримує A та у відповідь передає клієнту своє число B;
3. Генеруються секретні значення: C — на стороні клієнта, D — на стороні сервера;
4. Сторони виконують обчислення узгоджених значень: H = A^C mod B, G = A^D mod B;
5. Клієнт і сервер обмінюються обчисленими величинами H та G;
6. Кожна сторона обчислює спільний секретний ключ: K = G^C mod B, K = H^D mod B.

Після успішного узгодження ключ надсилається у головний виконуваний модуль програми, де він використовується для подальшого шифрування та дешифрування повідомлень. На цьому етапі активуються функції обміну повідомленнями, які використовують алгоритм AES зі згенерованим ключем. Для полегшення розуміння процесу на рис. 6 наведено його графічне (кольорове) та математичне (формульне) представлення.

Процес шифрування та відправлення повідомлень. Для надсилання повідомлень створено окремий клієнтський модуль, який постійно перебуває в режимі очікування нового введення. Перед відправленням текст повідомлення попередньо шифрується з використанням згенерованого спільного секретного ключа за алгоритмом AES. Після шифрування сформоване повідомлення передається до функції відправлення, де відкривається новий клієнтський сокет та здійснюється передача зашифрованих даних на вказану IP-адресу та порт отримувача у вигляді TCP-пакету. Після завершення передачі сокет закривається, а система повертається до очікування нового повідомлення.

Рисунок 6 – Візуально представлений процес обміну ключами

Процес отримання та дешифрування повідомлень. Для приймання вхідних даних запускається серверний модуль, який прослуховує визначений порт. Після надходження пакету з зашифрованим повідомленням він передається до функції дешифрування, де використовується попередньо узгоджений спільний ключ. У результаті отримане повідомлення відновлюється до відкритого тексту та відображається користувачу.

Таблиця 1 – Лістинг програми (серверна функція сканування)

```
def client_listen():
    data_dict ={}
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    client_socket.bind(('', server_port))

    while True:
        try:
            print("[*] Listening for broadcast messages...")
            print('\x1b[1A\x1b[2K', end='')
            data, addr = client_socket.recvfrom(1024)
            lens = (len(data_dict)+1)
            if lens != 0:
                for i in range(lens):
                    print('\x1b[1A\x1b[2K', end='')
            print("[*] Received message:", data.decode())
            print('\x1b[1A\x1b[2K', end='')
            ip, username, status = (data.decode()).split(' + ')
            if ip in data_dict:
                if data_dict[ip][0] == username :
                    if data_dict[ip][1] == status :
                        online_users_printing(data_dict)
                elif data_dict[ip] != [username, status] :
                    data_dict[ip][0] = username
                    data_dict[ip][1] = status
                    online_users_printing(data_dict)
            else:
```

```
            data_dict[ip] = [username, status]
            online_users_printing(data_dict)
        except:
            time.sleep(1)

def online_users_printing(users_dict):
    for i in users_dict :
        print(f"[*] User: {users_dict[i][0]} | Status: {users_dict[i][1]} | IP: {i}")
    print("[*] Input the last 3 digits IP of user that you want send data or 0 if you - receiver:")
```

Таблиця 2 – Лістинг програми (клієнтська функція сканування)

```
import socket

import netifaces

def get_local_ip():
    interfaces = netifaces.interfaces()
    for interface in interfaces:
        if interface != "lo":
            addresses = netifaces.ifaddresses(interface)
            if netifaces.AF_INET in addresses:
                ip_address = addresses[netifaces.AF_INET][0]["addr"]
                return ip_address

def server_broadcast(port, message):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    server_socket.sendto(message.encode(), ('192.168.0.255', port))

    server_socket.close()
```

Таблиця 3 – Лістинг програми (клієнтська частина функції генерації спільного ключа)

```
import socket
import threading
import signal
import sys

from Crypto.Random import get_random_bytes

def generate_private_value():
    return int.from_bytes(get_random_bytes(1), byteorder='big')

def compute_partial_key(base, private, modulus):
    return pow(base, private, modulus)

def key_exchange(target_host, key_length):
    target_port = 9990
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((target_host, target_port))
    print(f"[*] Key generation and exchange with {target_host}")
    client.sendall(f"L:{key_length}".encode())
    key_all = ''
    for i in range(0, key_length):
        try:
            a = generate_private_value()
```

```
        print(a)
        client.sendall(f"A:{a}".encode())

        b = client.recv(1024)
        if b.startswith(b"B:"):
            b = int((b.decode("utf-8")).split(":")[1])
            print(b)
        c = generate_private_value()

        print(c)

        result = compute_partial_key(a, c, b)
        print(result)
        client.sendall(f"H:{result}".encode())

        g_2 = client.recv(1024)
        if g_2.startswith(b"G:"):
            g = int((g_2.decode("utf-8")).split(":")[1])
            key = compute_partial_key(g, c, b)
            print(f"DH: Shared secret key: {key}")
            key_all = key_all + "/" + str(key)

    except:
        return False
  return key_all
```

Таблиця 4 – Лістинг програми (серверна частина функції генерації спільного ключа)

```
def DH_server(server_ip):
  try:
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((server_ip, 9990))
    server.listen(5)
    print(f"[*] Key exchange server started on {server_ip} [*]")
    key = ""
    data = []
    connection, address = server.accept()
    key_lenght = connection.recv(4096)
    for i in range(0,int((key_lenght.decode("utf-8")).split(":")[1])):
      print(f"[*] Connection accepted from {address[0]}:{address[1]} [*]")
  #     threading.Thread(target=handle_DH, args=(connection,)).start()
      key = key + "/" + str(handle_DH(connection))
  #     connection, address = server.accept()
    data.append(address[0])
    data.append(key)
    return data
  except:
    server.close()

def handle_DH(client_socket):

  request = client_socket.recv(4096)
  print(f'[*] Received: {request.decode("utf-8")}')

  if request.startswith(b"A:"):
    a = int((request.decode("utf-8")).split(":")[1])
    b = generate_private_value()
```

```
        client_socket.send(f"B:{b}".encode())

        h = int((client_socket.recv(4096)).decode("utf-8").split(":")[1])
        d = generate_private_value()
        g = compute_partial_key(a, d, b)
        key = compute_partial_key(h, d, b)
        print(f"DH_server: Shared secret key: {key}")
        client_socket.send(f"G:{g}".encode())
        return key
    else:
        client_socket.send(f"Connection refused".encode())
```

Таблиця 5 – Лістинг програми (функція шифрування)

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

def encr(message, key):
    print('\x1b[1A\x1b[2K', end='')
    print('\x1b[1A\x1b[2K', end='')
    print(f"[*] YOUR MESSAGE : {message}")
    print("[*] ENTER YOUR MESSAGE : ")
    key_ints = [int(x) for x in key.split('/')[1:]]
    dkey = bytes(key_ints)
    cipher = AES.new(dkey, AES.MODE_CFB)
    encrypted_data = cipher.encrypt(pad(message.encode(),AES.block_size))
    en_data=cipher.iv
    en_data += encrypted_data

    return en_data
```

Таблиця 6 – Лістинг програми (функція надсилання повідомлень)

```
import socket

target_port = 9983

def send_message(target_host,message):
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((target_host, target_port))
    client.sendall(message)
    client.close()
```

Таблиця 7 – Лістинг програми (функція сервера для отримування повідомлень)

```
def server_start(server_ip, key):
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((server_ip, 9983))
    server.listen(5)

    while True:
        connection, address = server.accept()
        client_handing = threading.Thread(target=handle_client, args=(connection, key, address[0]))
        client_handing.start()

def handle_client(client_socket, key, user2):
    while True:
        try:
            request = client_socket.recv(4096)
```

```
            message = decr(request, key)
            print('\x1b[1A\x1b[2K', end='')
            print(f'[*] {user2} : {message.decode("utf-8")}')
            print("[*] ENTER YOUR MESSAGE :")
            client_socket.send(b'ACK')
        except:
            return False
```

Таблиця 8 – Лістинг програми (функція дешифрування)

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

def decr(data, key):
    key_ints = [int(x) for x in key.split('/')[1:]]
    dkey = bytes(key_ints)
    iv=data[:AES.block_size]
    encrypted_data = data[AES.block_size:]
    decryption = AES.new(dkey, AES.MODE_CFB, iv=iv)
    decrypt_data=decryption.decrypt(encrypted_data)
    return decrypt_data
```

Таблиця 9 – Лістинг програми (основний файл main.py)

```
import threading
import signal
import sys
import time
import multiprocessing

from server import *
from scan import *
from Crypto.Random import get_random_bytes
from DH import key_exchange
from client import *
from encryption import encr
from auth import *

ready = "online"

def main():
    global ready
    global scan_f
    global client_listen_thread
    global Server_listen_thread
    username = auth()
    while True:
        if username == False :
            return False

        else:
            break

    signal.signal(signal.SIGINT, signal_handler)
    my_ipaddr = get_local_ip()
    scan_f = multiprocessing.Process(target=scan_function, args=(5, username, ready, ))
    scan_f.start()
    client_listen_thread = multiprocessing.Process(target=client_listen)
```

```
    client_listen_thread.start()
    target_user = input()
    key = ""
    if int(target_user) != 0 :
        scan_f.kill()
        client_listen_thread.kill()
        target_ip = my_ipaddr.split('.')
        target_ip[-1] = target_user
        target_ip = '.'.join(target_ip)
        key_length = input("Input key length : ")
        #DH_listen_thread = multiprocessing.Process(target=DH_server, args=(my_ipaddr,))
        #DH_listen_thread.start()
        key =""
        key = str(key_exchange(target_ip,int(key_length)))
#       DH_listen_thread.kill()
        print(key)
    else:
        client_listen_thread.kill()
        scan_f.kill()
        connection_data = DH_server(my_ipaddr)
        key = connection_data[1]
        target_ip = connection_data[0]
        print(key)

    Server_listen_thread = multiprocessing.Process(target=server_start, args=(my_ipaddr, key, ))
    Server_listen_thread.start()

    while True:
        message = input()
        message = encr(message, key)
        print(f"[*] ENCRYPTED MESSAGE :{message}")
        send_message(target_ip,message)

def scan_function(timeing, username, ready):
    while True:
        my_ipaddr = get_local_ip()
        status = f"{my_ipaddr} + {username} + {ready}"
        server_broadcast(9981, status)
        time.sleep(timeing)

def signal_handler(sig, frame):

    print("[*] Ctrl+C pressed. Stopping server...")
    scan_f.kill()
    client_listen_thread.kill()
    try:
        Server_listen_thread.kill()
    except:
#   DH_listen_thread.kill()
        sys.exit(0)

if __name__=="__main__":
    main()
```

Тестування утиліти. Для перевірки працездатності та оцінки безпечності утиліти було проведено тестування в локальній мережі за участю двох віртуальних машин, між якими здійснювався обмін даними. Також було виконано прослуховування мережевого трафіку з

метою аналізу переданих пакетів як у зашифрованому, так і в незашифрованому вигляді. Для дослідження того, як виглядають TCP-пакети, що передаються без використання шифрування, застосовано програму Wireshark (рис. 7.), яка дає змогу детально переглядати структуру мережевих пакетів та вміст переданих даних у відкритому вигляді.

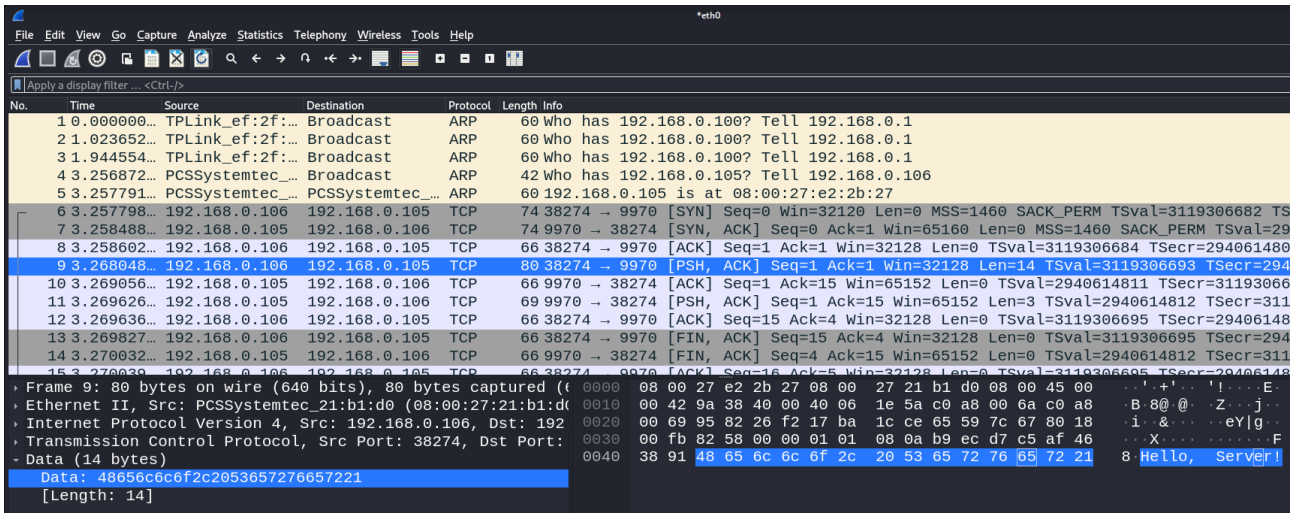

Рисунок 7 – Перехоплення нешифрованого зв’язку між двома хостами у локальній мережі

Як показано на рис. 7, у локальній мережі з хоста з IP-адресою 192.168.0.106 було відправлено повідомлення на хост 192.168.0.105 із текстом “Hello, server!”. Під час прослуховування мережі за допомогою програми Wireshark ці пакети були успішно перехоплені, і їхній зміст відображено у відкритому вигляді, оскільки передача даних здійснювалася без використання шифрування. Після цього було проведено тестування обміну повідомленнями вже з використанням шифрування. Для демонстрації роботи механізму захисту та порівняння з незашифрованим трафіком у програмі було додано виведення шифрованого повідомлення безпосередньо перед його відправленням (рис. 8). Це дозволяє наочно показати, що під час передачі через мережу вміст пакету є нерозбірливим та недоступним для читання без відповідного ключа дешифрування. Далі отримувач дешифрує повідомлення за допомогою спільного ключа і виводить його у термінал (рис. 9). Було також здійснено перехоплення цього з’єднання з подальшим аналізом переданих пакетів. У результаті перехоплення вдалося зафіксувати лише один пакет з корисним навантаженням, що відповідає моменту надсилання зашифрованого повідомлення “test message2” (рис. 10).

Рисунок 7 – Відправка та виведення зашифрованого повідомлення

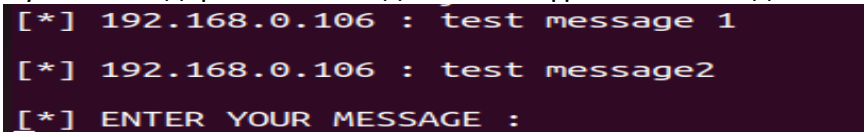

Рисунок 9 – Отримане і виведене дешифроване повідомлення

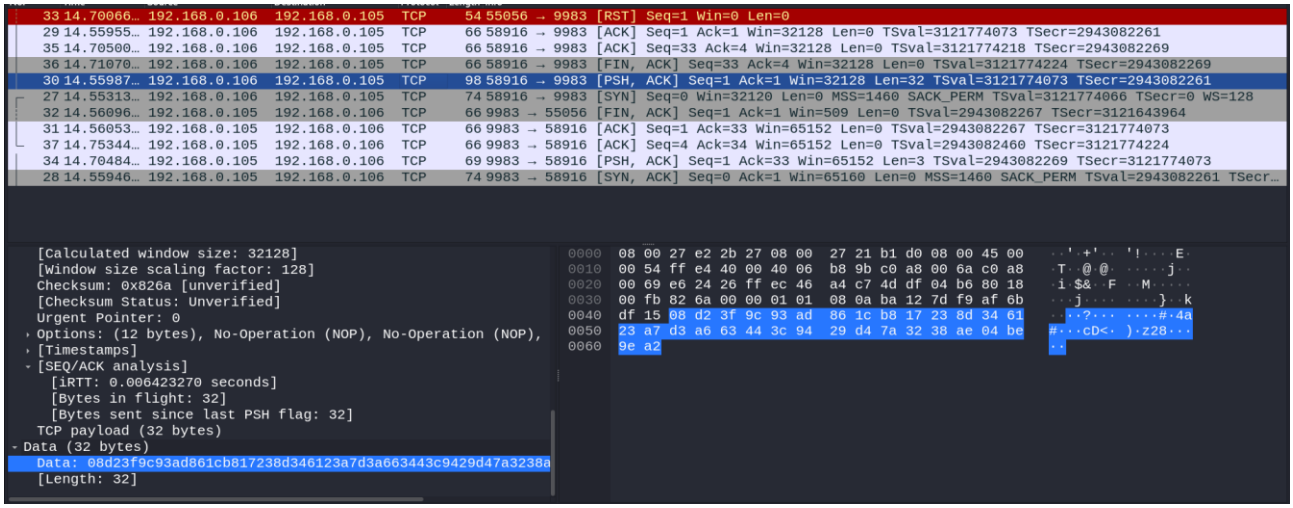

Рисунок 10 – Перехоплене шифроване повідомлення “test message2”

Як видно із перехопленого пакета, переглянути зміст повідомлення неможливо, оскільки воно зашифроване та не містить жодних відкритих даних. Це підтверджує, що утиліта коректно виконує поставлені перед нею завдання: забезпечує встановлення захищеного каналу зв’язку та передачу даних у локальній мережі.

Висновки

Було розглянуто основні методи наскрізного мережевого шифрування та створено утиліту, що реалізує функціонал месенджера у локальній мережі з використанням End-to-End Encryption для встановлення захищеного каналу передачі повідомлень.

Під час розробки утиліти застосовано симетричний алгоритм шифрування AES. Реалізація підтримує всі стандартні довжини ключів, доступні для AES, що дає користувачу можливість обирати розмір ключа та впливає на рівень безпеки передачі даних. Оскільки алгоритм є симетричним, один і той самий ключ використовується для шифрування та дешифрування повідомлень. Для генерації спільного ключа шифрування у мережі реалізовано алгоритм Діффі–Хеллмана. На відміну від класичного обміну готовими ключами, у цій утиліті використовується побайтове обчислення спільного ключа: випадково згенеровані байти перетворюються у двійкове, а потім у десяткове число, що використовується для формування ключа. Такий підхід підвищує стійкість алгоритму до атак на генератор псевдовипадкових чисел, оскільки передбачити значення кожного байта стає значно складніше.

Утиліта реалізована переважно з використанням стандартних бібліотек мови програмування Python, що робить її розробку простою, а виконання ефективним та надійним.

Фінансування

Це дослідження не отримало конкретної фінансової підтримки.

Конкуруючі інтереси

Автори заявляють, що у них немає конкуруючих інтересів.

Список використаних джерел

1. Jean Jarry, C., Rouphael, R., & Serror, J. (2019, March 4). From Julius Caesar to the blockchain: A brief history of cryptography. Variances. URL : https://www.ensae.org/fr/variances/article/from-julius-caesar-to-the-blockchain-a-brief-history-of-cryptography/4942

2. Sidhpurwala, H. (2023, January 12). *A brief history of cryptography*. URL : https://www.redhat.com/en/blog/brief-history-cryptography
3. Aumasson, J.-P. (2017). Serious Cryptography: A Practical Introduction to Modern Encryption. No Starch Press. URL : https://www.google.com.ua/books/edition/Serious_Cryptography/hLcrDwAAQBAJ?hl=ru&gbpv=0
4. Державна служба спеціального зв'язку та захисту інформації України. (2023, 12 липня). Що таке end-to-end шифрування, яке часто згадують, говорячи про месенджери? URL : https://www.cip.gov.ua/ua/news/sho-take-end-to-end-shifruvannya-yake-chasto-zgaduyut-govoryachi-pro-mesendzheri
5. Amazon Web Services. (n.d.). What Is SMTP? URL : https://aws.amazon.com/what-is/smtp/?nc1=h_ls
6. Краснов, О. В. (н. д.). Функції програми шифрування PGP та її застосування / Утиліти безпеки. Сучасна інформатика (сила практики в теорії). URL : https://alextexnok.blogspot.com/p/pgp.html
7. OpenPGP. (2024, 29 вересня). Про OpenPGP. URL : https://www.openpgp.org/about/
8. XMPP Standards Foundation. (n.d.). Про XMPP. URL : https://xmpp.org/about/
9. Goldberg, I., & команда розробників OTR. (н. д.). Off-the-Record Messaging [Сторінка проекту]. URL : https://otr.cypherpunks.ca/
10. Ehrenkret, M. (2023, 19 вересня). Квантова стійкість і протокол Signal. Signal. URL : https://signal.org/blog/pqxdh/
11. Національний інститут стандартів і технологій (NIST), Дворкін, М. Д., Баркер, Е., Нехватал, Дж., Фоті, Дж., Бассем, Л., Робак, Е., & Дрей молодший, Дж. (2001, 26 листопада). Advanced Encryption Standard (AES). Федеральні публікації з обробки інформації (FIPS 197). Національний інститут стандартів і технологій. https://doi.org/10.6028/NIST.FIPS.197
12. Srėbaliūtė, A. (2024, 28 травня). AES-шифрування: що це та як воно захищає ваші дані? NordLayer. URL : https://nordlayer.com/blog/aes-encryption/
13. GeeksforGeeks. (2025, 23 липня). Алгоритм RSA в криптографії. URL : https://www.geeksforgeeks.org/computer-networks/rsa-algorithm-cryptography/
14. Литвиненко, В. (2023, 21 квітня). Алгоритм шифрування RSA, види атак на нього. Реалізація мовою Python. DOU. URL : https://dou.ua/forums/topic/43026/
15. Rescorla, E. (1999, червень). Метод погодження ключа Диффі-Хеллмана (Diffie-Hellman Key Agreement Method). RFC 2631. Internet Engineering Task Force. URL : https://datatracker.ietf.org/doc/html/rfc2631
16. Cyber Witcher. (2023, 1 травня). Шпаргалка: MITM. HackYourMom. URL : https://hackyourmom.com/en/kibervijna/shpargalka-mitm/
17. Fortinet. (н. д.). Атака «людина-посередник» (Man-in-the-Middle): типи та приклади. URL : https://www.fortinet.com/resources/cyberglossary/man-in-the-middle-attack/
18. Imperva. (н.д.). ARP-спуфінг (ARP cache poisoning) — що це та як працює. URL : https://www.imperva.com/learn/application-security/arp-spoofing/
19. Кабінет Міністрів України. (н. д.). Що таке електронний цифровий підпис (ЕЦП)? URL : https://www.kmu.gov.ua/usi-pitannya-po-e-poslugam/sho-tak-elektronnij-cifrovij-pidpis-ecp

References

1. Serror, J., Jean Jarry, C., & Rouphael, R. (2019, March 4). From Julius Caesar to the blockchain: A brief history of cryptography. Variances. Retrieved from : https://www.ensae.org/fr/variances/article/from-julius-caesar-to-the-blockchain-a-brief-history-of-cryptography/4942

2. Sidhpurwala, H. (2023, January 12). A brief history of cryptography. Red Hat. Retrieved from : https://www.redhat.com/en/blog/brief-history-cryptography
3. Aumasson, J.-P. (2017). Serious Cryptography: A Practical Introduction to Modern Encryption. No Starch Press. Retrieved from : https://www.google.com.ua/books/edition/Serious_Cryptography/hLcrDwAAQBAJ?hl=ru&gbpv=0
4. State Service of Special Communications and Information Protection of Ukraine. (n.d.). What is end-to-end encryption often referred to when speaking about messengers? Retrieved from : https://www.cip.gov.ua/ua/news/sho-take-end-to-end-shifruvannya-yake-chasto-zgaduyut-govoryachi-pro-mesendzheri
5. Amazon Web Services. (n.d.). What Is SMTP? Retrieved from : https://aws.amazon.com/what-is/smtp/?nc1=h_ls
6. Krasnov, O. V. (n.d.). Functions of the PGP encryption program and its application / Security utilities. Modern Informatics (the power of practice in theory). Retrieved from : https://alextexnok.blogspot.com/p/pgp.html
7. OpenPGP. (2024, September 29). About OpenPGP. Retrieved from : https://www.openpgp.org/about/
8. XMPP Standards Foundation. (n.d.). About XMPP. Retrieved from : https://xmpp.org/about/
9. Goldberg, I., & the OTR Development Team. (n.d.). Off-the-Record Messaging [Project homepage]. Retrieved from : https://otr.cypherpunks.ca/
10. Ehrenkret, M. (2023, September 19). Quantum resistance and the Signal Protocol. Signal. Retrieved from : https://signal.org/blog/pqxdh/
11. National Institute of Standards and Technology (NIST), Dworkin, M. J., Barker, E., Nechvatal, J., Foti, J., Bassham, L., Roback, E., & Dray Jr., J. (2001, November 26). Advanced Encryption Standard (AES). Federal Information Processing Standards (FIPS 197). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.FIPS.197
12. Srėbaliūtė, A. (2024, May 28). AES encryption: What is it & how does it safeguard your data? NordLayer. Retrieved from : https://nordlayer.com/blog/aes-encryption/
13. GeeksforGeeks. (2025, July 23). RSA Algorithm in Cryptography. Retrieved from : https://www.geeksforgeeks.org/computer-networks/rsa-algorithm-cryptography/
14. Lytvynenko, V. (2023, April 21). RSA encryption algorithm, types of attacks on it. Implementation in Python. DOU. Retrieved from : https://dou.ua/forums/topic/43026/
15. Rescorla, E. (1999, June). Diffie-Hellman Key Agreement Method. RFC 2631. Internet Engineering Task Force. Retrieved from : https://datatracker.ietf.org/doc/html/rfc2631
16. Cyber Witcher. (2023, May 1). MITM cheat sheet. HackYourMom. Retrieved from : https://hackyourmom.com/en/kibervijna/shpargalka-mitm/
17. Fortinet. (n.d.). Man-in-the-Middle Attack: Types and Examples. Retrieved from : https://www.fortinet.com/resources/cyberglossary/man-in-the-middle-attack
18. Imperva. (n.d.). ARP Spoofing (ARP cache poisoning) — What it is and how it works. Retrieved from : https://www.imperva.com/learn/application-security/arp-spoofing/
19. Cabinet of Ministers of Ukraine. (n.d.). What is an electronic digital signature (EDS)? Retrieved from : https://www.kmu.gov.ua/usi-pitannya-po-e-poslugam/sho-tak-elektronnij-cifrovij-pidpis-ecp