Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
Факультет інформаційних технологій
Кафедра безпеки інформації та телекомунікацій

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеню бакалавра

студента *Олексюк Валерії Олександрівни*

академічної групи *125-20-1*

спеціальності *125 Кібербезпека*

спеціалізації[1]

за освітньо-професійною програмою *Кібербезпека*

на тему *Аналіз методів симетричного шифрування для захисту корпоративних даних*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	к.т.н., доц. Герасіна О.В.			
розділів:				
спеціальний	к.т.н., доц. Герасіна О.В.			
економічний	к.е.н., доц. Пілова Д.П.			
Рецензент				
Нормоконтролер	ст. викл. Мєшков В.І.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
безпеки інформації та телекомунікацій
_____ д.т.н., проф. Корнієнко В.І.

«_____»_____2024 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавра

студенту ***Олексюк Валерії Олександрівни*** **академічної групи** ***125-20-1***
(прізвище ім'я по-батькові) (шифр)

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

на тему ***Аналіз методів симетричного шифрування для захисту корпоративних даних***

затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 № 469-с

Розділ	Зміст	Термін виконання
Розділ 1	Проаналізувати стан безпеки у мережі та методи шифрування, зокрема RSA, для забезпечення конфіденційності та цілісності даних, а також розглянути цифрові підписи і управління ключами.	15.03.2024
Розділ 2	Визначити інструменти, використані при реалізації системи, та описати її можливості. Проаналізувати вибір алгоритмів шифрування, описати роботу серверної та клієнтської частин системи, зокрема зберігання даних та пряме з'єднання клієнтів. Описати готову програму.	10.05.2024
Розділ 3	Проаналізувати економічну доцільність розробки системи. Оцінити загальний збиток від можливих атак на підприємство. Визначити загальний економічний ефект від впровадження рекомендацій.	11.06.2024

Завдання видано _____ Олександра ГЕРАСІНА
(підпис керівника) (ім'я, прізвище)

Дата видачі: 15.01.2024р.

Дата подання до екзаменаційної комісії: 28.06.2023р.

Прийнято до виконання _____ Валерія ОЛЕКСЮК
(підпис студента) (ім'я, прізвище)

ЗМІСТ

РЕФЕРАТ

Пояснювальна записка: 75 с., 28 рис., 8 табл., 9 джерел.

Об'єкт дослідження: процес обміну повідомленнями в мережі Інтернет.

Мета роботи: розробка і впровадження системи обміну повідомленнями з використанням асиметричного і симетричного шифрування для забезпечення безпеки передачі даних.

Методи розробки: спостереження, порівняння, аналіз, опис.

У першому розділі було проаналізовано сучасні методи шифрування даних, стандарти та протоколи безпеки у сфері кібербезпеки, організаційне забезпечення обміну повідомленнями в мережі Інтернет, а також визначено актуальність впровадження наскрізне шифрування (E2EE).

У спеціальній частині було розроблено архітектуру системи обміну повідомленнями, впорядковано процеси генерації, зберігання та обміну криптографічними ключами між клієнтами, а також надано рекомендації щодо реалізації безпечного обміну повідомленнями.

В економічному розділі визначено економічну доцільність розробки та впровадження системи обміну повідомленнями. Проведено розрахунок капітальних (фіксованих) витрат, поточних (експлуатаційних) витрат та загального ефекту від впровадження системи.

Практичне значення роботи полягає у підвищенні ефективності та безпеки процесу обміну повідомленнями в мережі Інтернет завдяки розробці і впровадженню надійної системи з шифруванням.

Наукова новизна роботи полягає у використанні поєднання асиметричного та симетричного шифрування для забезпечення наскрізного шифрування в системі обміну повідомленнями.

АСИМЕТРИЧНЕ ШИФРУВАННЯ, ІНФОРМАЦІЙНА БЕЗПЕКА, КІБЕРБЕЗПЕКА, ОБМІН ПОВІДОМЛЕННЯМИ, СИМЕТРИЧНЕ ШИФРУВАННЯ, НАСКРІЗНЕ ШИФРУВАННЯ (E2EE), ШИФРУВАННЯ, AES, DES, RSA.

ABSTRACT

Explanatory Note: 75 pp., 28 pic., 8 table, 9 sources.

Research Object: The process of message exchange over the Internet.

Objective: To develop and implement a message exchange system utilizing both asymmetric and symmetric encryption to ensure data transmission security.

Development Methods: Observation, comparison, analysis, description.

In the first section, modern data encryption methods, security standards and protocols in the field of cybersecurity, and organizational support for message exchange over the Internet were analyzed, along with determining the relevance of implementing end-to-end encryption (E2EE).

In the specialized section, the architecture of the message exchange system was developed. The processes of generation, storage, and exchange of cryptographic keys among clients were organized, and recommendations for implementing secure message exchange were provided.

In the economic section, the economic feasibility of developing and implementing the message exchange system was determined. Calculations of capital (fixed) costs, operational (running) costs, and the overall effect of implementing the system were conducted.

The practical significance of the work lies in enhancing the efficiency and security of the message exchange process over the Internet, thanks to the development and implementation of a reliable encryption system.

The scientific novelty of the work is in the use of a combination of asymmetric and symmetric encryption to provide end-to-end encryption in the message exchange system.

Keywords: AES, ASYMMETRIC ENCRYPTION, CYBERSECURITY, DES, END-TO-END ENCRYPTION (E2EE), ENCRYPTION, INFORMATION SECURITY, MESSAGE EXCHANGE, RSA, SYMMETRIC ENCRYPTION.

СПИСОК УМОВНИХ СКОРОЧЕНЬ

АШ – асиметричне шифрування;

ВК – відкритий ключ;

ЗК – закритий ключ;

ІБ – інформаційна безпека;

ІН – ідентифікаційний номер;

КБ – кібербезпека;

КК – криптографічний ключ;

ПЗ – програмне забезпечення;

СШ – симетричне шифрування;

AES – Advanced Encryption Standard, стандарт шифрування даних;

E2EE – end to end encryption, наскрізне шифрування;

RSA – Rivest-Shamir-Adleman, алгоритм асиметричного шифрування;

SSL – Secure Sockets Layer, протокол захисту передачі даних;

TLS – Transport Layer Security, протокол безпеки транспортного шару.

ВСТУП

Важко переоцінити значущість комп'ютерних мереж у сучасному світі. Саме глобальна та широко розповсюджена мережа, під назвою Інтернет, щодня використовується мільйонами людей по всьому світу для вирішення різноманітних завдань та обміну даними. У Інтернеті люди можуть оплачувати послуги житлово-комунальних компаній, купувати подарунки для себе та друзів, відправляти накази банку на переказ коштів близьким. При обміні повідомленнями один з одним люди надсилають в мережу багато особистої інформації про себе, своїх дітей і роботу. Звичайно, нікому не хочеться, щоб його банківські операції чи особиста інформація потрапили в руки мошенників. Саме тому сьогодні питання безпеки комп'ютерних мереж та захисту інформації в них від сторонніх очей стоїть особливо гостро.

Метою цієї роботи було створення програмної системи, користувачі якої могли б обмінюватися повідомленнями між собою, не боячись компрометації передаваних та отриманих даних, а також, не залежати від використовуваної операційної системи, будь то Linux або Windows. Для досягнення цієї мети було реалізовано асиметричне та симетричне шифрування повідомлень, забезпечено підтримку p2p з'єднань, розроблено багатоплатформенний клієнт з графічним інтерфейсом, а також створено сервер для обробки та зберігання повідомлень.

Задача полягала у дослідженні теми, яка включала у себе аналіз сучасних методів шифрування, таких як AES та RSA, а також порівняння їх ефективності та швидкості роботи. Було розглянуто переваги та недоліки кожного з методів, що дозволило обрати найбільш оптимальні рішення для різних сценаріїв використання системи. Наприклад, асиметричне шифрування RSA використовується для обміну ключами, тоді як симетричне шифрування AES забезпечує високу швидкість передачі даних, що особливо важливо при обміні великими файлами. Також задачею було порівняння системи з існуючими рішеннями на ринку. На сьогоднішній день такі системи вже

існують і широко розповсюджені, як приклад можна навести Viber, Telegram, WhatsApp та інші. Однак багато з них мають закритий вихідний код, і перевірити, що ж насправді відбувається з вашими даними, не вдається. Розроблена система забезпечує певний рівень безпеки завдяки використанню комбінації асиметричного та симетричного шифрування, а також завдяки впровадженню технології наскрізного шифрування (E2EE). Крім того, підтримка p2p з'єднань дозволяє зменшити залежність від центрального сервера, що підвищує стійкість системи до атак і збоїв.

1. СТАН ПИТАННЯ. ПОСТАНОВКА ЗАДАЧІ

Безпека мережі — це досить складна тема, але її можна умовно поділити на кілька ключових аспектів: аутентифікація, конфіденційність та забезпечення цілісності зобов'язань. Аутентифікація визначає, чи має користувач право доступу до певної інформації, перш ніж йому буде надано доступ. Конфіденційність забезпечує, що інформація залишається недоступною для неавторизованих осіб. Контроль цілісності гарантує, що дані, отримані під час передачі, не були змінені ненавмисно або умисно. І нарешті, завдання забезпечення цілісності зобов'язань досягається за допомогою використання цифрових підписів.

Захист інформації в мережах базується на принципах криптографії. Тому варто обговорити це детальніше.

1.1 Основні принципи та поняття криптографії

Захист інформації в криптографії відбувається за допомогою процесу шифрування, який є оборотним перетворенням оригінального тексту в зашифрований текст за допомогою спеціального параметра, відомого як ключ. Традиційна криптографія використовує симетричні криптосистеми, де шифрування та розшифрування здійснюються за допомогою одного і того ж секретного ключа. Сучасна криптографія включає асиметричні криптосистеми, системи електронного підпису, хеш-функції, управління ключами, забезпечення конфіденційності інформації та багато іншого.

Повідомлення, що вимагають секретної передачі та називаються відкритим текстом, перетворюються в зашифрований текст на боці відправника за допомогою спеціальної функції шифрування та секретного ключа. Зашифрований текст потім передається до одержувача. Припускається, що по каналу зв'язку може спостерігатися певний зловмисник, який бачить і може компрометувати зашифровані повідомлення. Але для зловмисника

розшифровка повідомлень є складним завданням, а часто і неможливим без відповідного ключа. Існують два типи зловмисників: пасивний, який може лише спостерігати за каналом зв'язку, та активний, який може вставляти власні повідомлення або змінювати оригінальні перш, ніж вони досягнуть одержувача. На (рис. 1.1) показано модель шифрування-дешифрування і роль зловмисника в системі передачі даних.

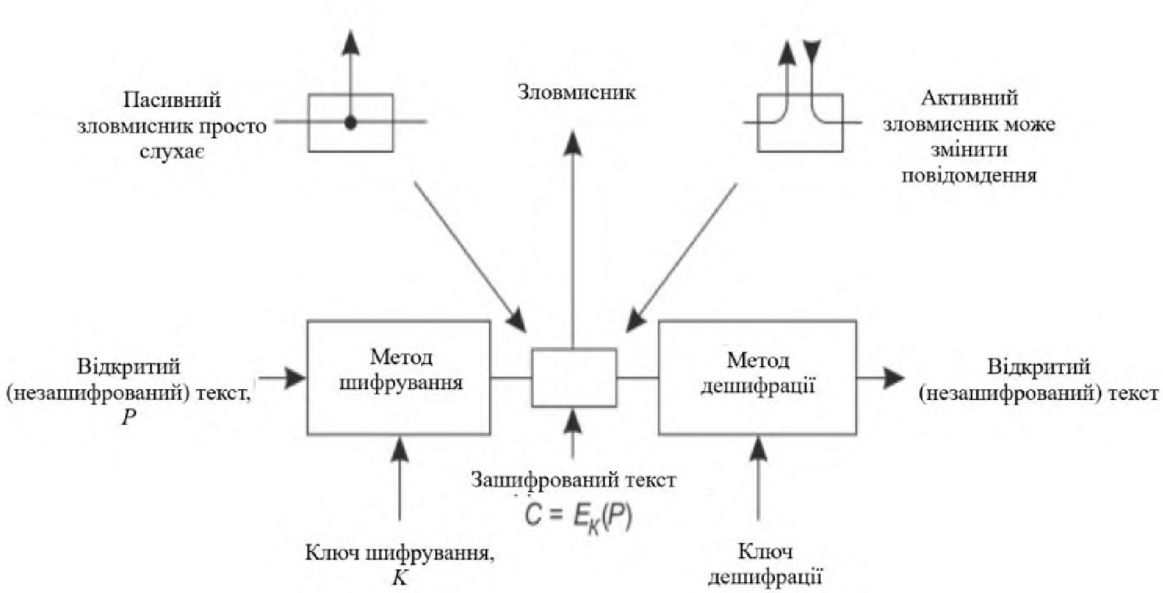

Рисунок 1.1 – Модель шифрування – дешифрування

На практиці, при компрометації одного методу шифрування, необхідно докласти величезні зусилля та ресурси для розробки іншого. Тому прийнято вважати, що жоден метод не є повністю секретним. Іншими словами, вважається, що зловмисник добре знає метод, який використовується для шифрування. Ідея, яка випливає з припущення, що злам можливий лише завдяки відомості про ексклюзивний ключ, називається принципом Керкгоффа, згідно з яким алгоритми шифрування мають бути загальнодоступними, а секретними є тільки ключі.

1.2 Шифрування

Будь-який сучасний шифр [1], що використовується на практиці, поєднує два основних методи: метод підстановки та метод перестановки.

Метод підстановки полягає у заміні кожного символу (або групи символів) вихідного тексту на інший символ (або групу символів). Важливо зазначити, що шифри на основі підстановок зберігають порядок слідування символів.

Шифри, у яких використовується метод перестановки, натомість змінюють лише порядок символів у вихідному тексті, але самі символи залишаються незмінними.

У традиційній криптографії використовувались прості алгоритми. Однак у сучасній криптографії метою є створення настільки складного і захищеного алгоритму, що без ключа неможливо отримати ніякої корисної інформації з зашифрованого тексту. Існують два класи алгоритмів шифрування — симетричні та з відкритим ключем.

1.2.1 Алгоритми шифрування з симетричним ключем

Клас алгоритмів з симетричним ключем [2] отримав свою назву тому, що ключ, який використовується для шифрування на боці відправника, співпадає з ключем розшифрування, використовуваним одержувачем. Основною проблемою таких алгоритмів є способи передачі секретного ключа між співрозмовниками, адже необхідно розподілити ключ між усіма учасниками обміну в абсолютно секретний спосіб.

Найвідоміші алгоритми шифрування з симетричним ключем включають:

— DES (Data Encryption Standard) — на даний момент вважається застарілим і не є безпечним;

— Triple-DES — трохи модифікований DES;

— AES (Advanced Encryption Standard) або Rijndael — найбільш розповсюджений алгоритм, який найчастіше використовується на практиці;

— Twofish — також широко використовується в промисловості.

1.2.2 Алгоритми шифрування з відкритим ключем

Діффі та Хеллман - дослідники з Стенфордського університету, які в 1976 році запропонували абсолютно нову криптосистему, в якій ключ шифрування і ключ дешифрування були різними [3]. У їх системі алгоритми шифрування (E) і дешифрування (D) повинні задовольняти трьом основним вимогам:

— D(E(P)) = P;

— вкрай складно вивести D з E;

— неможливо зламати, використовуючи випадковий відкритий текст.

З того часу було створено багато алгоритмів на основі концепції відкритих ключів. Алгоритм публічний, і не потрібно каналу безпеки для обміну ключами. Кожен користувач генерує пару ключів: один для шифрування, інший для розшифрування.

Кожен користувач публікує свій ключ шифрування (відкритий ключ) для загального доступу. Другий ключ (секретний ключ), що відповідає відкритому, зберігається в секреті.

Якщо користувач A хоче надіслати повідомлення користувачу B, він шифрує повідомлення відкритим ключем B. Коли користувач B отримує повідомлення, він розшифровує його за допомогою свого секретного ключа. Інші користувачі не можуть розшифрувати це повідомлення, оскільки лише B знає свій особистий ключ.

Найнадійнішим та найпоширенішим алгоритмом шифрування з відкритим ключем є алгоритм RSA, названий на честь його розробників Рівеста, Шаміра та Адлемана. Його математичний опис досить цікавий і не вимагає багато часу для осмислення, відрізняючись від інших алгоритмів, тому ми розглянемо його детальніше нижче.

1.2.2.1 Опис алгоритму RSA

Для розуміння алгоритму RSA [4] необхідно згадати деякі математичні твердження. Перш за все, нам потрібна функція Ейлера.

Функція Ейлера $\phi(n)$ — арифметична функція, яка дорівнює кількості натуральних чисел, менших за n та взаємно простих з n. Функція Ейлера має наступні важливі властивості:

— якщо n та m — взаємно прості числа, то

$$\phi(nm)=\phi(n)\cdot\phi(m); \tag{1.1}$$

— якщо p — просте число, то

$$\phi(p)=p-1. \tag{1.2}$$

Також важливі наступні теореми:

Теорема 1: Мала теорема Ферма

Якщо p — просте число та a — ціле число, таке що НСД (a,p) = 1, то

$$a^{p-1} \equiv 1(mod\ p), \tag{1.3}$$

Теорема 2: Китайська теорема про залишки

Нехай $m1, m2, \dots, mk$ — цілі, взаємно прості числа,

$$M = \Pi_{i=1}^{k} m_i, \tag{1.4}$$

Тоді для будь-якого набору цілих чисел $a1, a2, \dots, ak$ існує унікальне рішення системи порівнянь:

$$\begin{cases} x \equiv a_1(mod\ m_1) \\ x \equiv ak(mod\ m_k)' \end{cases} \tag{1.5}$$

Рішення системи єдине за модулем M та відповідає рівнянню:

$$x \equiv \sum_{i=1}^{k} a_i b_i c_i (mod\ M), \tag{1.6}$$

Де

$$b_i = \frac{1}{m_i} \circ \left(\prod\nolimits_{j=1}^{k} m_j \right) = \frac{M}{m_i}, \tag{1.7}$$

Та

$$c_1 \equiv b_i^{-1}(mod\ m_i), \tag{1.8}$$

На цьому завершується необхідна математична база. Тепер перейдемо до процесу шифрування. Кожен абонент вибирає свою пару ключів. Для цього він генерує два великих простих числа p і q, та обчислює їх добуток N:

$$N = p \times q, \tag{1.9}$$

Далі вибирається випадкове число e, взаємно просте зі значенням функції Ейлера від числа N, відповідно до вищезазначених властивостей, тобто:

$$\phi(N)=(p-1)\times(q-1), \tag{1.10}$$

Тоді знаходиться число d за умови:

$$e\times d\equiv 1(\mathrm{mod}\phi(N)), \tag{1.11}$$

За такими НСД(e,$\phi(N)$ = 1, це число існує, але воно єдине. Пара (N,e) є відкритим ключем та публікується для загального доступу. Пара (N,d) є секретним ключем і зберігається в таємниці. Для розшифровки достатньо знати секретний ключ.

Для того, щоб зашифрований текст міг бути переведений у числову форму будь-яким способом, результатом чого є прийом від певного великого числа.

Після того, як отримане число поділено на частини, кожна з них має бути числом у діапазоні [0, N – 1]. Процес шифрування однаковий для кожної частини. Таким чином, ми можемо вважати, що частина вихідного тексту представлена числом x, де

$$0\leq x<N, \tag{1.12}$$

Користувач А, відправляючи повідомлення користувачу В, вибирає з відкритого каталогу пару (N, e) користувача В і виконує шифрування повідомлення за формулою:

$$y = x^e (mod\, N), \tag{1.13}$$

Користувач В відновлює вихідний текст, використовуючи формулу:

$$x = y^d (mod\, N), \tag{1.14}$$

Давайте покажемо, що обчислене значення дійсно дорівнює вихідному тексту.

Маємо:

$$y^d = (x^e)^d = x^{ed}(mod\, N), \tag{1.15}$$

Оскільки e·d≡1(modϕ(N)), це можна представити у вигляді:

$$e \cdot d = 1 + k \cdot \phi(N), \tag{1.16}$$

для деякого цілого числа k, що означає:

$$x^{ed} = x^{1+k_\varphi(N)}(mod\, N) = x^{1+k(p-1)(q-1)}(mod\, p), \tag{1.17}$$

Розглянемо два випадки:

— якщо x ділиться на p, то:

$$x^{1+k(p-1)(q-1)} \equiv 0(mod\, p), \tag{1.18}$$

— якщо x не ділиться на p, то:

$$x^{1+k(p-1)(q-1)} \equiv x \cdot (x^{p-1})^{k(q-1)}(mod\, p), \tag{1.19}$$

Згідно з малою теоремою Ферма:

$$(x^{p-1})^{k(q-1)}(\mathrm{mod}\, p) = 1^{k(q-1)}(\mathrm{mod}\, p) = 1(\mathrm{mod}\, p), \tag{1.20}$$

Маємо:

$$y^d = x^{ed} = x \cdot (x^{p-1})^{k(q-1)}\, mod\, p = x \cdot 1\, mod\, p = x\, mod\, p, \tag{1.21}$$

Аналогічні розрахунки можна провести, помінявши p з q, тобто це порівняння виконується для числа p і для числа q, і, відповідно, за китайською теоремою про залишки, це порівняння виконується і для добутку чисел p і q. Відповідно:

$$y^d = x^{ed} = x\, mod\, N, \tag{1.22}$$

Що й треба було довести.

1.3 Цифрові підписи

У деяких ситуаціях люди можуть намагатися відмовитися від своїх раніше прийнятих зобов'язань. У сучасному світі зобов'язання, які людина бере на себе, часто фіксуються на папері, а потім людина може заперечувати ці

зобов'язання, що ставить під загрозу репутацію, оскільки важко довести угоду з ними. Комп'ютерні системи також потребують в аналогічного механізму.

Необхідна система, яка б у переписці між двома користувачами гарантувала виконання наступних умов:

— отримувач має можливість перевірити справжність відправника;

— відправник не може заперечувати відправлене повідомлення після його надсилання;

— отримувач не має можливості модифікувати підписане повідомлення.

Цифрові підписи [5] саме і є такою системою. Також, як і шифри з відкритим ключем, цифрові підписи діляться на дві категорії – підписи з відкритим і закритим ключем.

1.3.1 Підписи з симетричним ключем

Методи підписів з симетричним ключем включають створення центрального органу довіри (ЦОД), якому всі довіряють. Кожен користувач вибирає секретний ключ та ділиться ним із ЦОД. Коли Наталка хоче надіслати Миколі текст P, вона формує повідомлення, зашифроване ключем Наталки, KA, таке як $KA(B,RA,t,P)$, де B — ідентифікатор Миколи, RA — випадкове число, вибране Наталкою, t — часова мітка, що підтверджує актуальність повідомлення. Потім Наталка надсилає це повідомлення в ЦОД. ЦОД розшифровує повідомлення Наталки, після чого формує своє повідомлення, зашифроване ключем Миколи, $KB(A,RA,t,P)$, і надсилає його Миколі. $KC(A,t,P)$, $KC(A,t,P)$ — повідомлення зашифроване ключем ЦОД є видимим підписом ЦОД.

1.3.2 Підписи з відкритим ключем

Основна проблема підписів з симетричним ключем полягає в тому, що кожен повинен довіряти певному Центральному Органу, через який проходять всі повідомлення. Було б краще, якби нікому не доводилося довіряти.

Шифрування з відкритим ключем може допомогти в цьому. Для цього ми вимагаємо, щоб алгоритми шифрування і дешифрування мали властивість $E(D(P))=P$. Алгоритм RSA володіє цією властивістю. В такій схемі Наталка відправляє Миколі підписане повідомлення $EB(DA(P))$ (текст, зашифрований спочатку закритим ключем Наталки, а потім відкритим ключем Миколи), отримавши таке повідомлення Микола розшифровує його своїм закритим ключем, а потім відкритим ключем Наталки. При цьому підпис буде $DA(P)$, оскільки DA відомо тільки Наталці.

1.4 Профілювання повідомлень

Методи цифрових підписів часто використовують поєднання двох відмінних функцій: автентифікації та секретності. Зазвичай вимагається лише автентифікація.

Система автентифікації, яка не вимагає шифрування, заснована на використанні хеш-функцій. Хеш-функції — це функції, призначені для створення стислого представлення повідомлення з довільного тексту. В результаті виходить стисле значення, зване хешем або дайджестом повідомлення (Message Digest, MD), яке має наступні властивості:

— за заданим текстом P легко обчислити хеш-функцію MD(P);

— з дайджеста MD(P) практично неможливо відновити оригінальний текст P;

— практично неможливо знайти текст P', такий щоб MD(P) = MD(P');

— зміна хоча б одного біта вхідного тексту призводить до зовсім іншого результату хеш-функції.

Профілювання повідомлень за допомогою частини відкритого тексту виконується швидше, ніж шифрування цілого повідомлення. Тому

профілювання можна використовувати для прискорення цифрових підписів. Розглянемо, як профілювання повідомлень може допомогти у випадку цифрового підпису з симетричним ключем: ЦОД замість повного повідомлення KB(A, RA, t, P) відправляє KB(A, RA, t, P, KC(A, t, MD(P))). Використання профілю дозволяє заощадити час на шифрування та зменшити витрати на транспортування та зберігання.

Профілювання повідомлень також може використовуватися для гарантування секретності повідомлення під час його передачі по мережах з відкритим ключем. Наприклад, Наталка спочатку вираховує профіль свого повідомлення P = MD(P), а потім шифрує (підписує) профіль повідомлення — DA(MD(P)) і надсилає його Миколі поряд з P, DA(MD(P)) — це відкритий текст та підписаний профіль. Найвідоміші алгоритми цифрового підпису включають MD5 (Message Digest Algorithm 5) — нестійкий алгоритм, широко поширений та SHA-1 (Secure Hash Algorithm 1), який також почав втрачати на надійності та замінюється на SHA-2.

1.5 Управління ключами

Використання асиметричної криптографії дозволяє користувачам обмінюватися конфіденційними даними, не встановлюючи перед цим обміном секретних ключів, підписувати свої повідомлення електронним підписом, не довіряючи їх третім особам, а також перевіряти, чи не були повідомлення змінені.

Але як двоє незнайомців обмінюються відкритим ключем перед початком розмови? Просто залишити відкритий ключ на якомусь сайті в Інтернеті не є безпечним, оскільки коли один з співрозмовників робить запит на сайт для отримання відкритого ключа, зловмисники можуть замінити оригінальний ключ на вірусний сайт на свій, а потім перехопити і змінити повідомлення. Тому потрібен механізм, який дозволив би безпечно обмінюватися відкритими ключами.

1.5.1 Сертифікати

Таким механізмом стали організації, зайняті сертифікацією відкритих ключів, які називаються центрами сертифікації (ЦС). Тепер, якщо хтось хоче налагодити безпечне з'єднання з незнайомою особою, він повинен звернутися в ЦС зі своїм відкритим ключем і зареєструвати його. ЦС видає сертифікат, аналогічний показаному на рисунку 2, з хеш SHA-1, який буде підписаний закритим ключем цього центру. Завданням цього сертифіката є зв'язування відкритого ключа з його власником. Сам по собі цей сертифікат не захищається і не зберігається в таємниці.

Також, як і раніше, щоб двоє незнайомців почали спілкуватися, їм потрібно виводити свої сертифікати в загальний доступ. Однак тепер, якщо зловмисник намагається підробити сертифікат або відкритий ключ, це не спрацює, тому що хеш цього сертифіката при перевірці не співпадає з хешем, обчисленим по відкритому ключу центру сертифікації в блоку підпису (рис. 1.2).

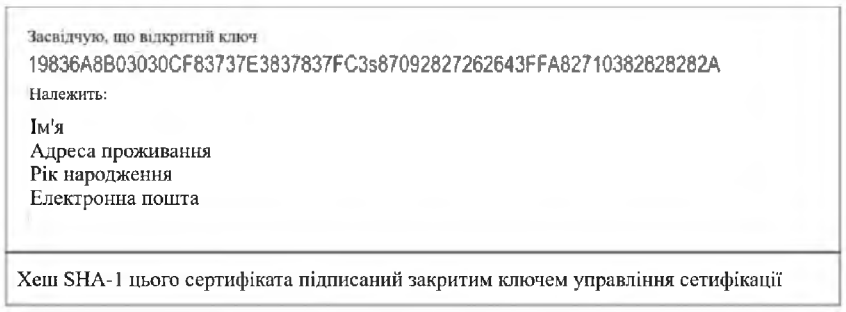

Рисунок 1.2 – Зразковий вид сертифіката

Насправді це цілком природно, що для отримання сертифіката вам не доведеться нікуди йти, оскільки було розроблено спеціальний стандарт сертифікатів X.509, який широко поширений в інтернеті.

Один Центр Сертифікації на весь світ, звісно, недостатньо. З цієї причини був створений метод сертифікації відкритих ключів під назвою PKI (Public Key Infrastructure – інфраструктура систем з відкритими ключами).

Одним із типів PKI є ієрархія управлінь, представлена на малюнку 3. На малюнку 3 зображено три рівня, однак їх може бути як більше, так і менше. Центр сертифікації верхнього рівня ми будемо називати центральним управлінням (ЦУ), воно сертифікує управління другого рівня, назвемо їх Регіональними відділами (РВ). Регіональні відділи, в свою чергу, сертифікують справжні Управління сертифікації, які видають клієнтам сертифікати стандарту X509 (рис. 1.3).

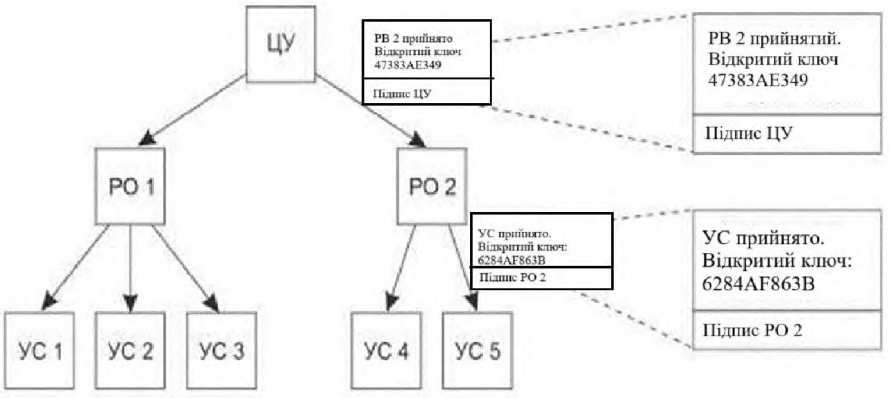

Рисунок 1.3 – Ієрархія управлінь

Тепер, коли хтось, наприклад, Наталка, хоче перевірити сертифікат свого співрозмовника Миколи, вона запитує в нього сам сертифікат, а також ланцюжок сертифікатів управлінь вищих рівнів. Такий ланцюжок називається ланцюжком довіри або шляхом сертифіката. Таким чином Наталка може перевірити всі організації, які її цікавлять, за допомогою цього ланцюжка. Але як перевірити відкритий ключ Центрального управління? Припускається, що відкритий ключ Центрального управління відомий усім, зазвичай ключі різних таких управлінь «вшиті» всередину браузерів та інших клієнтських додатків.

1.6 Захист інформації у всесвітньому павутинні. SSL та TLS.

На самому початку існування інтернету ніхто особливо не турбувався про безпеку переданих даних, однак, коли люди усвідомили всі його переваги і почали розробляти різні додатки для нього, наприклад, стали

використовувати інтернет для проведення фінансових операцій, стало зрозуміло, що передані дані потребують захисту. У 1995 році громадськості була представлена нова система безпеки під назвою SSL (Secure Sockets Layer — протокол захищених сокетів) [6]. Відповідне програмне забезпечення, як і сам протокол SSL, сьогодні широко використовується.

Цей протокол встановлює між двома користувачами безпечне з'єднання, дозволяючи:

— узгодити параметри, які будуть використовуватись для передачі даних;

— підтвердити особистості користувачів;

— організувати між користувачами таємне спілкування;

— забезпечити цілісність і захист переданих даних.

Після того, як безпечне з'єднання встановлене, SSL займається підтримкою компресії та шифрування переданих даних. Існує кілька версій SSL, останньою і найнадійнішою є SSLv3. Залежно від версії, SSL може мати різні додаткові функції, включаючи наявність або відсутність компресії, той чи інший алгоритм шифрування, а також деякі речі, пов'язані з обмеженнями експорту в криптографії.

SSL складається з двох підпротоколів, один з яких призначений для встановлення захищеного з'єднання, а другий — для використання цього з'єднання. Почнемо з розгляду питання встановлення з'єднання, візьмемо, наприклад, Наталку та Миколу. Робота підпротоколу, що займається цим, зображена на рисунку 1.4. Все починається з повідомлення, в якому Наталка відправляє Миколі запит на встановлення з'єднання. У ньому вона вказує використовувану версію SSL, а також свої переваги щодо алгоритмів компресії та шифрування. У цьому повідомленні також міститься число Наталки RA, називане нонс (одноразовий код), який буде використано згодом. Далі вступає Микола. У відповідному повідомленні він вибирає один з алгоритмів, запропонованих Наталкою, та додає свій нонс RB. У наступному своєму повідомленні він відправляє свій сертифікат з відкритим ключем і ланцюжок

довіри цього сертифіката, якщо це необхідно. Після того, як Микола виконав свою частину протоколу, він повідомляє, що настала черга Наталки. Наталка у відповідь генерує 384-бітний підготовчий ключ, шифрує його своїм відкритим ключем і відправляє його Миколі. Справжній ключ сеансу буде обчислений на основі нонсів обох сторін і підготовчого ключа. Після отримання цього повідомлення Наталка і Микола обчислюють сеансовий ключ. Для цього Наталка просить Миколу змінити шифр, а також повідомляє, що вона вважає підпротокол встановлення з'єднання завершеним, після чого Микола погоджується з нею.

Другий підпротокол, який використовується для передачі даних, представлений на рисунку 1.5. Повідомлення, що надходять від користувача, розбиваються на одиниці даних розміром до 16 Кбайт і, за необхідності, компресуються незалежно один від одного. Потім до стислого тексту додається закритий ключ, а результат хешується за попередньо узгодженим алгоритмом. Хеш додається до кожного фрагменту у вигляді MAC (Message Authentication Code — код аутентифікації повідомлення). Цей стислий фрагмент разом з MAC кодується узгодженим алгоритмом з симетричним ключем і передається по каналу зв'язку одержувачу.

У 1996 році протокол SSL був відправлений на стандартизацію. Результатом став стандарт TLS (Transport Layer Security — протокол безпеки транспортного рівня) [6]. TLS був заснований на основі SSLv3. При створенні стандарту TLS в SSL було внесено не так багато змін, але цих змін вистачило, щоб зробити SSLv3 та TLS несумісними. В результаті, для усунення проблем з сумісністю протоколів, багато клієнтських додатків використовують обидва протоколи, і при необхідності TLS перетворюється назад у SSL, таке рішення називається SSL/TLS.

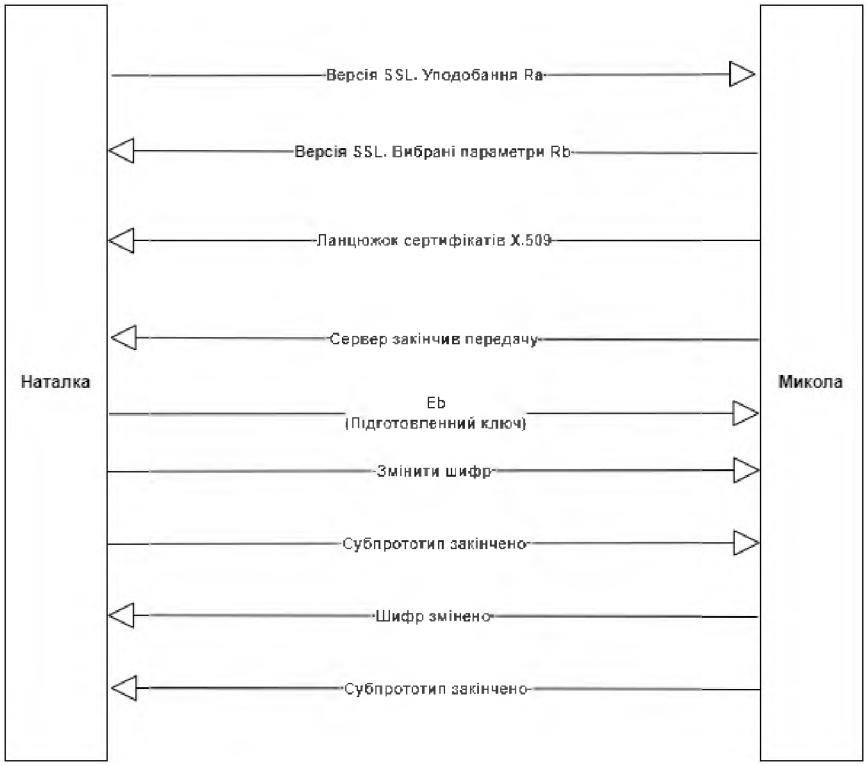

Рисунок 1.4 – Встановлення SSL з'єднання

Рисунок 1.5 – передача даних через SSL

1.7 Висновки до розділу

У цьому розділі розглянуто ключові аспекти безпеки у мережі, зокрема методи шифрування, цифрові підписи, профілювання повідомлень, управління ключами та захист інформації у всесвітньому павутинні за допомогою протоколів SSL та TLS.

Проаналізовано алгоритми шифрування з симетричним ключем, такі як AES, та алгоритми шифрування з відкритим ключем, включаючи RSA. Симетричне шифрування характеризується високою швидкістю та ефективністю, що робить його ідеальним для шифрування великих обсягів даних. Однак, воно вимагає безпечного обміну ключами, що може бути складним завданням. Асиметричне шифрування, навпаки, забезпечує високий рівень безпеки завдяки використанню пари ключів (відкритого та закритого), але є більш ресурсномістким. У розділі детально описано алгоритм RSA, який широко використовується для захищеного обміну ключами в асиметричних системах.

Цифрові підписи є важливим елементом забезпечення цілісності та автентичності повідомлень. Розглянуто підписи з симетричним ключем, які забезпечують швидке створення та перевірку підписів, але мають обмежену застосовність через необхідність спільного секретного ключа. Підписи з відкритим ключем, такі як DSA та RSA, забезпечують високий рівень безпеки і не вимагають попереднього обміну секретами, що робить їх більш гнучкими у використанні.

Профілювання повідомлень дозволяє аналізувати та класифікувати трафік для виявлення аномалій та загроз. Це важливий інструмент для моніторингу безпеки мережі та виявлення потенційних атак.

Управління ключами є критичним аспектом криптографічних систем. Розглянуто процеси генерації, зберігання та розповсюдження ключів, а також роль сертифікатів у забезпеченні довіри. Сертифікати дозволяють підтвердити автентичність ключів та забезпечують додатковий рівень безпеки.

Нарешті, розглянуто протоколи SSL та TLS, які забезпечують захищений обмін даними у всесвітньому павутинні. Ці протоколи використовують комбінацію симетричного та асиметричного шифрування для забезпечення конфіденційності та цілісності даних під час їх передачі через Інтернет.

Розглянуті у цьому розділі методи та технології є фундаментальними для забезпечення безпеки в мережі. Шифрування, цифрові підписи, управління ключами та використання протоколів SSL/TLS є невід'ємними компонентами сучасних систем захисту інформації. Впровадження цих технологій дозволяє забезпечити високий рівень конфіденційності, цілісності та автентичності даних, що є критичним для захисту корпоративних клієнтів та забезпечення довіри до інформаційних систем. Використання поєднання симетричного та асиметричного шифрування, профілювання повідомлень та управління ключами створює надійну основу для захищеного обміну повідомленнями та ефективного управління інформаційною безпекою.

2. СПЕЦІАЛЬНИЙ РОЗДІЛ

Для виконання роботи була реалізована система обміну повідомленнями в мережі Інтернет, до якої входять серверний додаток, клієнтський додаток, а також деякі додаткові модулі, необхідні для їх роботи. Дана система є кросплатформеною, тобто може працювати на різних операційних системах: Windows, macOS та системах, заснованих на ядрі Linux, таких як Android (з деякими обмеженнями), Ubuntu, Linux Mint, Raspbian та інших. Також ця система дозволяє передавати та зберігати повідомлення підписаними, зашифрованими та з гарантією цілісності, таким чином, доступ до них матимуть лише ті люди, для яких вони призначені, а будь-які зміни, яким вони можуть бути піддані зловмисниками, будуть виявлені. Архітектура системи в загальному вигляді представлена на рисунку 2.1.

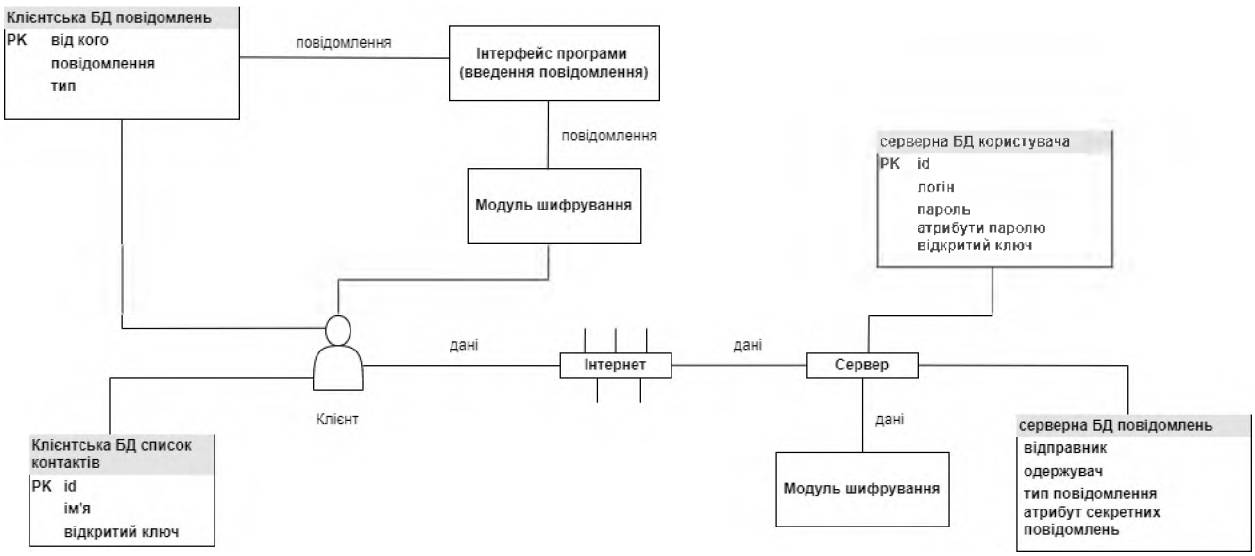

Рисунок 2.1 – Архітектура розробленої системи

2.1 Інструменти, використані при реалізації

Для написання основного програмного коду системи було обрано мову програмування Python [7], графічний інтерфейс додатка було реалізовано за допомогою модуля tkinter, що постачається у складі стандартної бібліотеки Python. Зберігання даних додатка, таких як повідомлення, список користувачів та інших, здійснюється в базі даних SQLite, перевага була віддана саме їй, оскільки це легковагова, швидка, надійна та широко розповсюджена база даних, відмінно підходяща як для настільних комп'ютерів, так і для мобільних пристроїв. Для реалізації криптографічних функцій, таких як цифрові підписи, хешування, симетричне та асиметричне шифрування, використовувалася бібліотека cryptography.fernet з відкритим вихідним кодом. Написання коду відбувалося в середовищі розробки VS Code від компанії Microsoft. Для перегляду вмісту баз даних використовувалося розширення SQLite Manager для браузера Google Chrome. Пояснювальні ілюстрації та схеми були намальовані за допомогою програм GIMP та draw.io.

2.2 Можливості реалізованої системи

Можливості серверної частини даної системи досить прості і в основному зводяться до обслуговування клієнтських запитів, таких як аутентифікація, запити до бази даних та пересилання повідомлень від одного користувача до іншого. Тому перейдемо відразу до розгляду клієнтської частини системи.

Щоб скористатися клієнтським додатком, користувач спочатку повинен пройти процес реєстрації, або, якщо він уже зареєстрований, процес аутентифікації, таким чином забезпечується гарантія того, що користувач є тим, за кого він себе видає. При реєстрації користувач повинен вибрати собі ім'я (логін) з тих, що ще не були використані, тобто не містяться в базі даних користувачів, що зберігається на сервері, і яке буде відображатися іншим

користувачам при спілкуванні з ним, а також за допомогою якого інші користувачі зможуть додати цю особу до свого списку контактів.

Щоб додати особу до списку своїх контактів, потрібно просто ввести його логін у спеціальне поле введення та натиснути на кнопку «Додати друга», після чого він відобразиться в списку контактів.

Після того, як користувач пройшов процес аутентифікації і знайшов людей, з якими він планує почати спілкування, він вибирає цікаву йому особу зі свого списку контактів і спосіб передачі повідомлень.

Передача повідомлень може здійснюватися одним із чотирьох способів:

— передача не зашифрованих повідомлень від одного користувача до іншого через сервер;

— передача зашифрованих повідомлень від одного користувача до іншого через сервер. Для цього перед передачею повідомлення потрібно спочатку встановити секретний ключ, натиснувши на кнопку «встановити секретний ключ»;

— передача не зашифрованих повідомлень безпосередньо між клієнтами, виключаючи проміжне ланка у вигляді сервера. Для цього перед передачею повідомлення потрібно спочатку встановити з'єднання безпосередньо, натиснувши на кнопку «встановити p2p з'єднання»;

— передача зашифрованих повідомлень безпосередньо між клієнтами, виключаючи проміжне ланка у вигляді сервера. Для цього потрібно встановити і секретний ключ, і пряме з'єднання, якщо вони не були встановлені раніше.

Схеми передачі повідомлень для способів 1 та 2 представлені на рисунку 2.2, а для способів 3 та 4 — на рисунку 2.3.

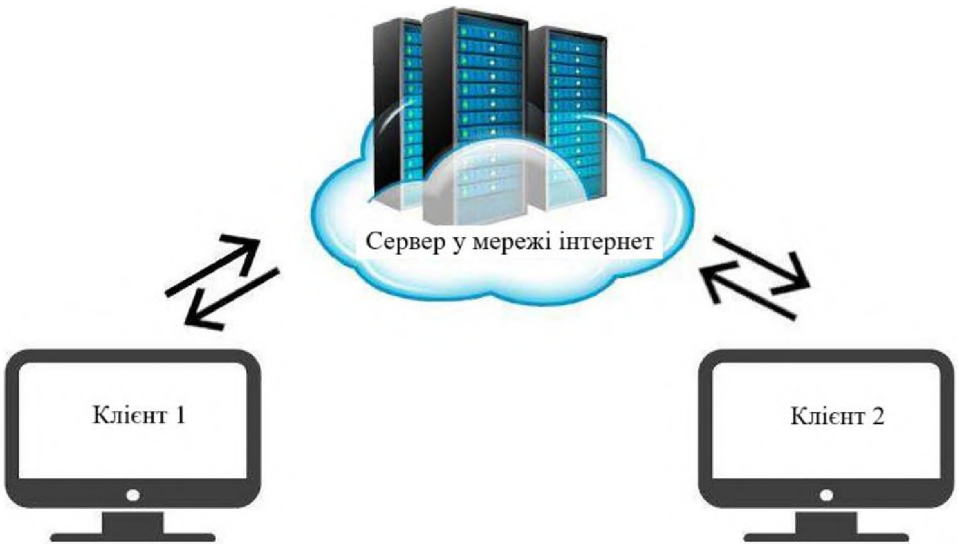

Рисунок 2.2 – Схема передачі даних способами 1 та 2

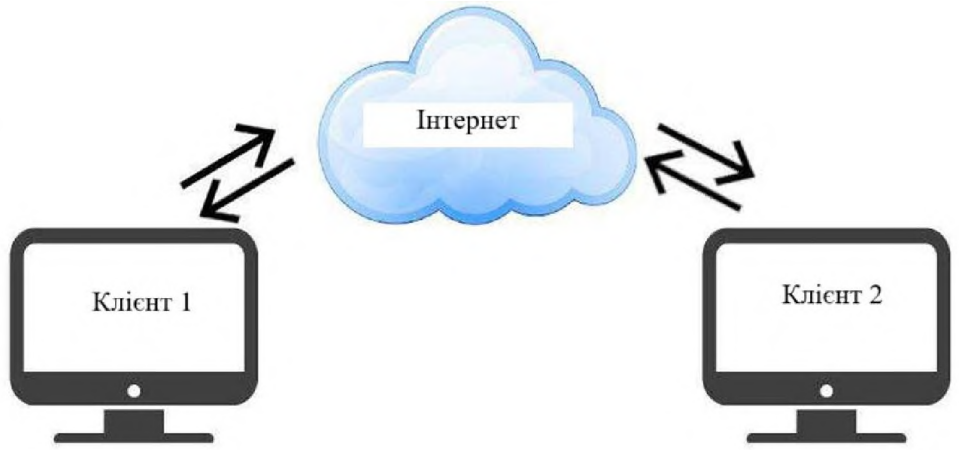

Рисунок 2.3 – Схема передачі даних способами 3 та 4

При передачі повідомлень способами 1 та 3 немає жодної гарантії, що повідомлення, відправлені користувачем, не будуть перехоплені та прочитані сторонніми людьми, однак підмінити або якимось чином змінити передане між користувачами повідомлення без їх відома не вийде, оскільки повідомлення буде позначене як пошкоджене, ці способи підходять для звичайного спілкування без передачі якої-небудь таємної інформації, особистої чи службової, паролів і так далі.

При передачі повідомлень способами 2 та 4 нікому не вдасться не тільки якимось чином непомітно пошкодити повідомлення, але також і прочитати його або дізнатися якусь важливу інформацію, оскільки повідомлення будуть зашифровані при передачі від першого клієнта і розшифровані лише при отриманні другим клієнтом (навіть сервер, у випадку передачі способом 2, не зможе розшифрувати передане повідомлення). Ці способи підходять для передачі інформації будь-якого типу, однак будуть також і додаткові часові витрати на шифрування та розшифрування цієї інформації. Цими чотирма способами передачі даних гарантується цілісність повідомлень, і, у випадку зашифрованої передачі, їх таємниця.

Зазначимо, що будь-яким із цих способів можна передавати не тільки повідомлення, але й будь-які файли.

В будь-який момент часу користувач може вийти зі свого акаунта або зовсім видалити його з бази даних сервера.

Також клієнтський додаток здатний працювати і без графічного інтерфейсу, тобто в консольному режимі, що менш зручно, але може бути корисним при відсутності графічного менеджера, зазвичай, таке може бути на системах Linux, зібраних самостійно. Консольний режим роботи є на даний момент єдиним можливим для системи Android, саме про це обмеження згадувалося раніше.

2.3 Про вибір алгоритмів та способи шифрування

Як уже було зазначено раніше, найбільш часто використовуваними та поширеними алгоритмами шифрування для симетричних і асиметричних криптосистем є AES та RSA відповідно, тому саме вони були обрані для шифрування даних у реалізованій системі.

Однак алгоритм RSA використовується лише для того, щоб зашифрувати та передати секретний ключ, який буде використовуватися алгоритмом AES (про це докладніше буде розказано в пункті 2.5). Так було зроблено тому, що

RSA працює значно повільніше, ніж AES. Значна різниця в швидкості роботи обумовлена тим, що алгоритм AES у своїй роботі використовує переважно перестановки та зсуви, які не є обчислювально затратними операціями, в той час як RSA використовує у своїй роботі піднесення до ступеня, множення та взяття залишку за певним модулем великих чисел, що навіть для сучасних комп'ютерів є досить складним завданням. Результати порівняння швидкостей для шифрування ста повідомлень та ста файлів наведені в таблиці 2.1.

Таблиця 2.1 – Порівняння швидкостей роботи алгоритмів AES та RSA

Алгоритм\тип повідомлення	Текст (10 – 20 Байт)	Файл (10-20 КБайт)
RSA (повний час, сек)	2.3835	354.8178
RSA (средній час, сек)	0.0238	3.5481
AES (повний час, сек)	0.0648	0.0717
AES (средній час, сек)	0.0006	0.0007

З цієї таблиці видно, що RSA працює у десятки разів повільніше, ніж AES для невеликих текстових повідомлень, і в тисячі разів повільніше для відносно малих файлів. На практиці ж файли, які передаються по мережі, важать десятки або навіть сотні мегабайт, і шифрувати їх алгоритмом RSA — погана ідея.

2.4 Опис роботи серверної частини системи

Для початку опишемо принцип роботи сервера. Як тільки сервер запустився вперше, передбачається, що він завжди буде активним. При першому запуску сервер проводить початкову ініціалізацію, тобто створює дві бази даних та всі необхідні таблиці в них, одну для зберігання клієнтських повідомлень, іншу для зберігання списку клієнтів. Після ініціалізації сервер переходить у режим очікування підключення клієнтів.

Підключення клієнта відбувається наступним чином: як тільки сервер приймає вхідне підключення, він встановлює з клієнтом SSL-з'єднання, щоб передати йому ключ симетричного шифрування, який складається з 32 випадково згенерованих байтів і буде використовуватися в подальшому. Після передачі ключа SSL-з'єднання закривається, і канал зв'язку знову стає незахищеним. Далі сервер створює новий потік, у якому буде обслуговуватися тільки що підключений клієнт, а поточний потік продовжує очікувати нових вхідних підключень.

Обслуговуючий клієнта потік завжди перебуває в стані очікування якогось повідомлення від клієнта. Повідомлення може бути одного з двох видів: повідомлення, призначене для пересилання іншому користувачеві, або повідомлення, призначене безпосередньо серверу, тобто якась команда. Першим повідомленням від клієнта сервер очікує отримати команду – запит аутентифікації, якщо він отримує якісь інші повідомлення, то вони просто ігноруються до тих пір, поки клієнт не буде аутентифікований. Після успішної аутентифікації клієнта сервер виконуватиме будь-які його допустимі команди і пересилатиме повідомлення іншим клієнтам.

Перелічимо список допустимих команд, які виконує сервер та їх опис.

Реєстрація – клієнт надіслав дані для реєстрації, логін і пароль (а також деякі інші дані, про які ми поговоримо, коли будемо розглядати роботу клієнта) будуть занесені в базу даних на сервері, якщо не станеться якоїсь помилки, наприклад, якщо клієнт спробує зареєструватися з логіном, який вже зайнятий, після чого клієнту призначається ідентифікаційний номер і відправляється йому зворотним повідомленням. Логін і пароль завжди надходять у зашифрованому вигляді.

Вхід – клієнт надіслав логін і пароль, сервер перевіряє, чи є така пара в його базі даних, і в залежності від результату перевірки клієнт може отримати наступні відповіді:

— логін, з яким намагається увійти у систему клієнт, не знайдений в базі даних клієнтів;

— пароль, з яким намагається увійти у систему клієнт, не відповідає логіну;

— повідомлення про успіх, у випадку, якщо пара логін/пароль була знайдена сервером у своїй базі даних.

Видалення акаунта – клієнт надіслав запит на видалення свого акаунта з бази даних клієнтів. Сервер знаходить логін поточного клієнта і видаляє його з бази даних разом з усією інформацією про цього клієнта, однак історія його повідомлень не буде видалена, оскільки вона може знадобитися іншим користувачам, з якими він вів діалог.

Вихід з акаунта – клієнт вийшов із свого акаунта, сервер знову очікує повідомлення з запитом автентифікації, інші повідомлення ігноруються.

Пошук іншого клієнта за логіном – клієнт надіслав логін клієнта, якого хоче додати до списку своїх контактів. Якщо сервер знаходить такий логін у своїй базі даних, то відповідним повідомленням відправляє ідентифікаційний номер, що відповідає цьому логіну. Якщо сервер такого логіна не знайшов, то у відповідь буде відправлено повідомлення про помилку.

Отримання історії повідомлень з іншими користувачами – як тільки користувач успішно пройшов автентифікацію, сервер висилає йому історію його повідомлень з іншими користувачами, збережену у базі даних.

Створення прямого зв'язку між клієнтами – клієнт надіслав запит на встановлення прямого зв'язку, щоб спілкуватися, обходячи сервер. Про встановлення такого зв'язку буде розказано трохи пізніше.

Обмін ключами шифрування між клієнтами – клієнт надіслав два однакові симетричні ключі шифрування, зашифровані двома різними способами, для чого це потрібно буде пояснено при детальному розгляді роботи користувацького додатка. Сервер збереже ці два ключа в базу даних і призначить їм певний номер (ідентифікатор сесії), за яким їх можна буде знайти в подальшому. Один із ключів буде перенаправлений клієнту - отримувачу.

Завершення зв'язку з боку клієнта – клієнт закрив додаток. Сервер відразу перерве зв'язок зі своєї сторони.

При отриманні від клієнта повідомлення, призначеного іншому клієнту, сервер перевіряє, чи є це повідомлення зашифрованим, і якщо так, то просто пересилає його отримувачу, не застосовуючи до нього жодних дій, оскільки йому невідомий ключ, яким його зашифрували та підписали (цей ключ відомий лише відправнику та отримувачу і відрізняється від тих, які сервер висилав їм при підключенні), якщо ж повідомлення не є зашифрованим, то воно обов'язково є підписаним тим ключем, який сервер вислав клієнту на початку обслуговування, оскільки клієнту-отримувачу цей ключ невідомий, то сервер перевіряє, що повідомлення не було пошкоджено на шляху від клієнта-відправника до сервера і, якщо все в порядку, то сервер пере підписує повідомлення ключем, відомим йому та клієнту-отримувачу, після чого відправляє повідомлення цьому клієнту.

Наглядна схема роботи серверного додатка представлена у вигляді блок-схем на рисунках 2.4 та 2.5.

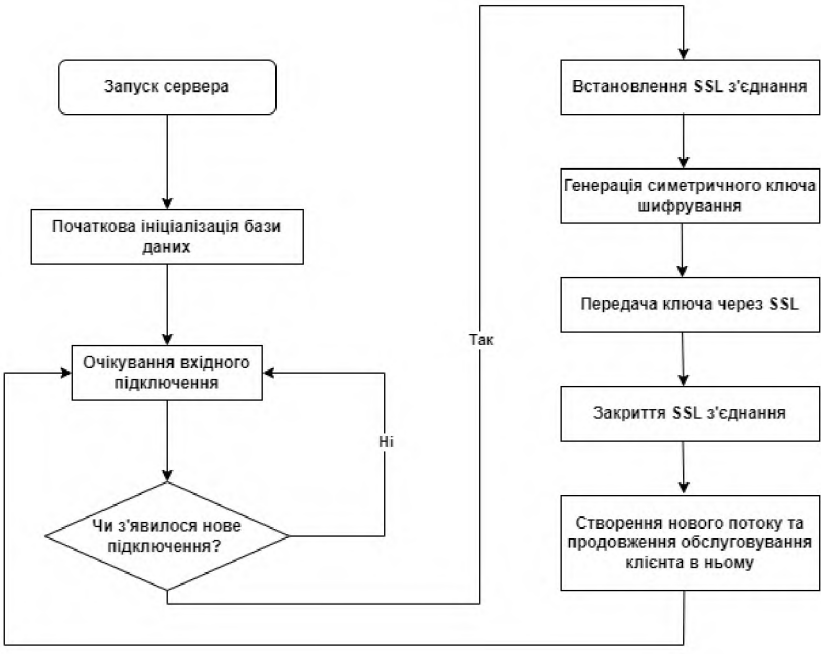

Рисунок 2.4 – Блок-схема роботи серверної програми

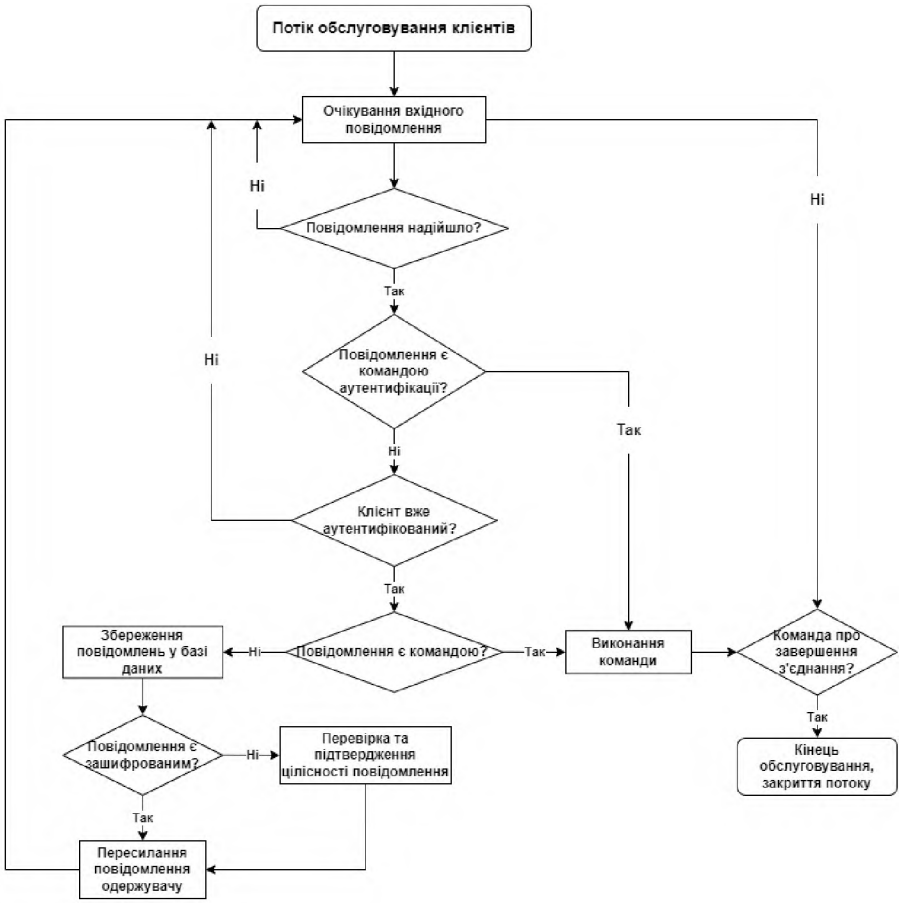

Рисунок 2.5 – Блок-схема роботи потоку обслуговування клієнта на сервері

2.4.1 Зберігання даних на сервері

Тепер поговоримо детальніше про те, як і які дані сервер зберігає у своїх базах даних. Всю необхідну інформацію сервер зберігає не в первісному вигляді, а перетворює в байти. Це було зроблено для того, щоб, по-перше, скоротити використовуваний дисковий простір, а по-друге, тому що більшість даних потрібно пересилати клієнтам, тобто їх все одно довелося б перетворювати у набір байтів.

Для початку розглянемо відомості, які сервер зберігає про клієнта. Для зберігання даних про клієнта сервер використовує базу даних, що складається з однієї таблиці, яка називається user_list і містить наступні поля:

— uid – ідентифікаційний номер, присвоюваний сервером клієнту в момент реєстрації;

— login – логін клієнта у вигляді набору байтів;

— hashed_password – хеш клієнтського пароля і спеціальної добавки, званої сіллю, що це таке і навіщо це потрібно буде пояснено далі;

— saltl – частина «солі», приписувана паролю зліва у момент хешування;

— saltr – частина «солі», приписувана паролю справа у момент хешування;

— public_key – відкритий ключ користувача для асиметричного шифрування.

Список записів у таблиці може виглядати як показано на рисунку 2.6 (зображення розділено на 2 частини ліву і праву):

uid	login	hashed_password
1	117,115,101,114,49	54,66,248,94,167,16,2,26,91,54,171,54,150,80,91,85,26,193,35,67,147,48,42,179,109,185,40,88,200,242,19,209
2	117,115,101,114,50	145,73,117,127,244,208,148,147,232,176,58,158,228,185,33,173,179,56,46,185,138,209,132,241,166,227,211,78,194,160,177,170
3	117,115,101,114,51	124,242,121,191,195,32,147,29,194,20,81,19,251,11,65,41,232,65,40,29,113,91,43,249,203,39,33,131,66,246,227,111
4	117,115,101,114,52	21,40,197,23,147,72,138,175,194,145,169,127,190,88,176,212,76,166,254,242,157,75,6,247,210,252,52,153,130,170,154,245
5	97,100,109,105,110	2,46,34,25,119,160,211,248,121,161,149,25,76,50,120,194,226,112,76,10,54,65,140,215,64,77,198,177,163,32,64,203

saltl	saltr	public_key
202,72,249,217,7,83,195,152	78,167,205,116,197,137,122,213	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,4
43,132,172,156,233,96,106,113	66,159,83,152,190,74,3,239	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,4
255,39,17,78,29,78,44,56	100,109,185,186,153,216,129,240	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,4
192,152,23,159,141,112,158,33	177,18,117,117,72,110,238,63	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,4
204,89,253,4,45,133,246,8	128,67,218,235,62,243,207,44	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,4

Рисунок 2.6 – Приклад бази даних паролів ліва частина (згори) та права (знизу).

Розберемося, навіщо потрібно «солити» паролі перед їх поміщенням у базу даних. Зберігання паролів у вихідному вигляді просто нерозумно, адже якщо хтось зможе отримати доступ до бази даних на сервері, то він зможе безперешкодно отримати доступ і до будь-якого йому цікавого акаунту. Щоб цьому завадити, зазвичай зберігають не сам пароль, а лише його хеш, тоді той, хто отримає доступ до бази даних, уже не зможе так просто отримати доступ до потрібного акаунту, оскільки доведеться перебирати всі можливі паролі та

порівнювати їх хеші з тим, що зберігається в базі даних, що може потребувати дуже багато часу.

Однак на сьогоднішній день цей метод також не вважається достатньо безпечним, оскільки в інтернеті існують так звані «веселкові таблиці». Веселкова таблиця — це така таблиця, в якій зберігаються паролі, що відповідають певним хешам. Таким чином, маючи хеш пароля і виконавши його пошук по веселковій таблиці, можна отримати список паролів, яким відповідає цей хеш при різних алгоритмах хешування. В такому випадку може допомогти так звана «сіль». Сіль — це додатковий рядок, згенерований випадковим чином, який приписується до пароля і хешується разом з ним. З отриманого таким чином хешу за допомогою веселкової таблиці пароль уже не відновиш. Знаючи сіль і вихідний хеш, зловмисник змушений на повний перебір пароля.

Тепер розглянемо, в якому вигляді сервер зберігає різні клієнтські повідомлення. Зашифровані та не зашифровані повідомлення зберігаються в різних таблицях. Історія повідомлень клієнтів з ідентифікаційними номерами a і b зберігається в таблиці з назвою message_history_a_b, або secret_message_history_a_b — якщо це зашифровані повідомлення.

Таблиця не зашифрованих повідомлень містить наступні поля:

— sender_id – ідентифікаційний номер відправника;

— receiver_id - ідентифікаційний номер отримувача;

— message – повідомлення;

— message_type – тип повідомлення (текстове або файл).

В таблиці зашифрованих повідомлень зберігаються всі ті самі поля, що і для не зашифрованих повідомлень, але крім них, є ще кілька інших полів:

— session_id – номер секретної сесії;

— message_tag, message_nonce – дані, необхідні для розшифровки повідомлення.

Кожного разу при передачі секретних повідомлень клієнти повинні домовитися про відомий тільки їм ключ, за допомогою якого вони будуть шифрувати свої дані. При встановленні клієнтами нового ключа сервер отримує два зашифрованих ключа, які він повинен зберегти, для того щоб клієнти в майбутньому могли розшифрувати ці повідомлення. Більше того, всі повідомлення, які передаються між користувачами за допомогою цього ключа, повинні бути якимось чином асоційовані з ним. Поле session_id саме й вказує, якому ключу відповідають поточні повідомлення. Щоб отримати ключі, знаючи session_id повідомлень, серверу необхідно звернутися до ще однієї таблиці, названої encrypted_session_keys_a_b, де a і b – ідентифікаційні номера користувачів. У цій таблиці містяться такі поля:

— session_id – номер секретної сесії;

— key_encrypted_by_first_id, key_encrypted_by_second_id – ключі для поточної сесії.

Приклад повідомлення, що міститься в таблиці не секретних повідомлень, представлено на рисунку 2.7.

sender_id	receiver_id	message	message_type
2	1	103,104,98,100,116,110	1

Рисунок 2.7 – Приклад запису в таблиці не секретних повідомлень

Вся необхідна інформація, що стосується зберігання даних сервером, була нами розглянута. Тепер перейдемо до завершальної частини розгляду роботи сервера та поговоримо про встановлення зв'язку безпосередньо між клієнтами.

2.4.2 Пряме з’єднання клієнтів

Стандарт IP версії 4 нараховує всього трохи більше чотирьох мільярдів інтернет-адрес. У сучасному світі цих адрес уже давно не вистачає для всіх. Проблему нестачі адрес має вирішити стандарт IP версії 6, однак впровадження цього стандарту відбувається досить повільно, а доступ до інтернету людям потрібен тут і зараз. Тому було придумано NAT (Network Address Translation – перетворення мережевих адрес) [8].

Локальні мережі всередині будинку, організації або міста можуть використовувати величезну кількість IP-адрес. NAT дозволяє цю величезну кількість адрес перетворити в одну. Перетворення адреси методом NAT може здійснюватися майже будь-яким маршрутизуючим пристроєм, наприклад, роутером. Найбільш популярним є SNAT, суть механізму якого полягає у заміні адреси джерела при проходженні інтернет-пакета в один бік і зворотній заміні адреси призначення у відповідному пакеті. Окрім адрес джерело/призначення також можуть замінюватися номери портів джерела і призначення.

Отримуючи пакет від локального комп'ютера, роутер дивиться на IP-адресу призначення. Якщо це локальна адреса, то пакет пересилається іншому локальному комп'ютеру. Якщо ні, то пакет треба переслати назовні в інтернет. Але ж зворотною адресою в пакеті вказана локальна адреса комп'ютера, яка з інтернету буде недоступна. Тому роутер «на льоту» транслює (замінює) зворотню IP-адресу пакета на свою зовнішню (видиму з інтернету) IP-адресу і змінює номер порта (щоб розрізняти відповідні пакети, адресовані різним локальним комп'ютерам). Комбінацію, потрібну для зворотної підстановки, роутер зберігає у себе в тимчасовій таблиці. Через деякий час після того, як клієнт і сервер закінчать обмінюватися пакетами, роутер стирає у себе в таблиці відповідний запис.

Приклад роботи пристрою NAT показаний на рисунку 2.8.

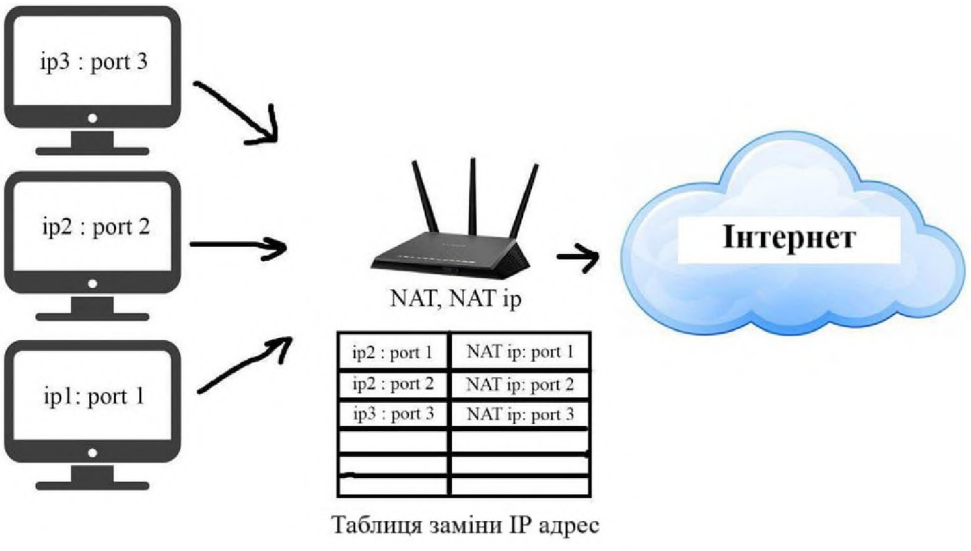

Рисунок 2.8 – Схема роботи пристрою NAT

Незважаючи на всю користь NAT, він також вносить деякі обмеження у вигляді того, що з мережі, яка знаходиться за межами NAT, немає можливості підключитися до пристрою, який знаходиться «за» NAT, не вдаючись до деяких хитрощів. Найчастіше ефективним і добре описаним методом встановлення прямого зв'язку є Hole punching (пробивання отворів). «Пробивання» TCP-отвору дозволяє двом клієнтам встановити прямий одноранговий TCP-сеанс за допомогою відомого сервера, навіть якщо обидва клієнти знаходяться за NAT. Hole punching передбачає, що два клієнти, A і B, вже встановили зв'язок з сервером S. Коли клієнт надсилає запит на встановлення прямого зв'язку з іншим клієнтом, сервер S записує дві адреси для цього клієнта: пару (IP-адреса, порт), яку клієнт вважає, що використовує для спілкування з S, тобто його адресу всередині локальної мережі, і пару (IP-адреса, порт), яку сервер спостерігає при отриманні клієнтського повідомлення. Ми називаємо першу пару приватною адресою клієнта, а другу — публічною адресою клієнта. Якщо клієнт не знаходиться за NAT, то його приватна і публічна адреси мають співпадати.

Основна практична проблема для додатків, які бажають реалізувати Hole punching за протоколом TCP, - це не проблема протоколу, а проблема інтерфейсу сокетів. Оскільки стандартний інтерфейс сокетів Берклі був

розроблений навколо парадигми клієнт/сервер, він дозволяє використовувати сокет протоколу TCP для ініціації вихідного зв'язку або для прослуховування вхідних зв'язків, але не обидві дії одразу. Крім того, сокети TCP зазвичай мають однозначне відповідність номерам портів TCP на локальному комп'ютері: після того, як додаток прив'язує один сокет до певного локального порту TCP, спроба прив'язати другий сокет до того ж порту TCP закінчується невдачею.

Однак для того, щоб «пробити отвір», нам потрібно використовувати один локальний порт TCP, щоб прослуховувати вхідні TCP-з'єднання та одночасно ініціювати кілька вихідних TCP-з'єднань. На щастя, усі основні операційні системи підтримують спеціальний параметр сокета TCP, який зазвичай називається SO_REUSEADDR, який дозволяє додатку зв'язувати кілька сокетів з однією й тією ж локальною кінцевою точкою, якщо цей параметр встановлений на всіх задіяних сокетах. Системи BSD ввели опцію SO_REUSEPORT, яка контролює повторне використання порта окремо від повторного використання адреси; в таких системах мають бути встановлені обидві ці опції.

Припустимо, що клієнт A хоче встановити зв'язок з клієнтом B. Ми припускаємо, що A і B вже встановили зв'язки з відомим сервером S. Протокол hole punching працює наступним чином:

— клієнт A використовує свій активний сеанс з S, щоб попросити S допомогти підключитися до B;

— S висилає клієнту A публічну та приватну адресу B, і в той же час відправляє B публічну та приватну адресу A;

— з тих же портів, які A і B використовували для зв'язку з S, кожен з них починає виконувати спроби з'єднання з приватними та публічними адресами другого, одночасно прослуховуючи вхідні з'єднання;

— A і B чекають успішної спроби вихідного з'єднання і/або появи вхідного з'єднання. Якщо одна з спроб вихідного з'єднання завершується невдачею через мережеву помилку, комп'ютер просто повторює цю спробу з'єднання

після невеликої затримки (наприклад, однієї секунди), до досягнення максимальної кількості спроб;

— коли зв'язок встановлено, клієнти автентифікують один одного, щоб переконатися, що вони підключені до того, кого хотіли. Якщо автентифікація не вдалася, клієнти закривають це з'єднання і продовжують чекати, поки інші спроби з'єднання досягнуть успіху. Клієнти використовують для спілкування перше успішно автентифіковане з'єднання, отримане в результаті цього процесу;

— розглянемо сценарій загального випадку, в якому клієнти A і B знаходяться за різними NAT, як показано на малюнку. Спроби з'єднання A і B з приватними адресами один одного проваляться або призведуть до підключення до неправильного хосту. Важливо, щоб додатки автентифікували свої однорангові сеанси через можливість помилкового підключення до випадкового хосту в локальній мережі, який може мати той же приватний IP-адрес, що і потрібний хост на віддаленому комп'ютері.

Процес роботи алгоритму hole punching представлений на рисунках 2.9, 2.10 та 2.11.

Спроби вихідного підключення клієнтів до публічних адрес один одного призводять до того, що відповідні NAT відкривають нові «діри», забезпечуючи прямий зв'язок TCP між A та B. Якщо NAT є «дружнім» до методу hole punching, то між клієнтами автоматично створюється нове однорангове з'єднання. Якщо перша спроба з'єднання з B з боку A досягає пристрою NAT клієнта B до того, як перша спроба з'єднання з A з боку B досягає свого NAT, тоді NAT клієнта B може інтерпретувати з'єднання з боку A як спробу незапрошеного вхідного з'єднання і відхилити його. Однак з'єднання з боку B при цьому має пройти, оскільки пристрій NAT клієнта A бачить це з'єднання як частину вихідного сеансу, який було розпочато його клієнтом.

На жаль, деякі пристрої NAT не допускають з'єднання таким чином, тому встановити прямий зв'язок між клієнтами вдасться не завжди.

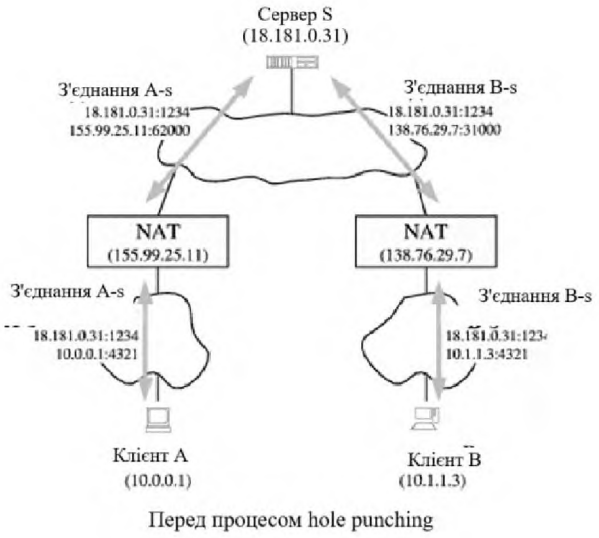

Рисунок 2.9 – Схема з'єднань перед початком процесу hole punching

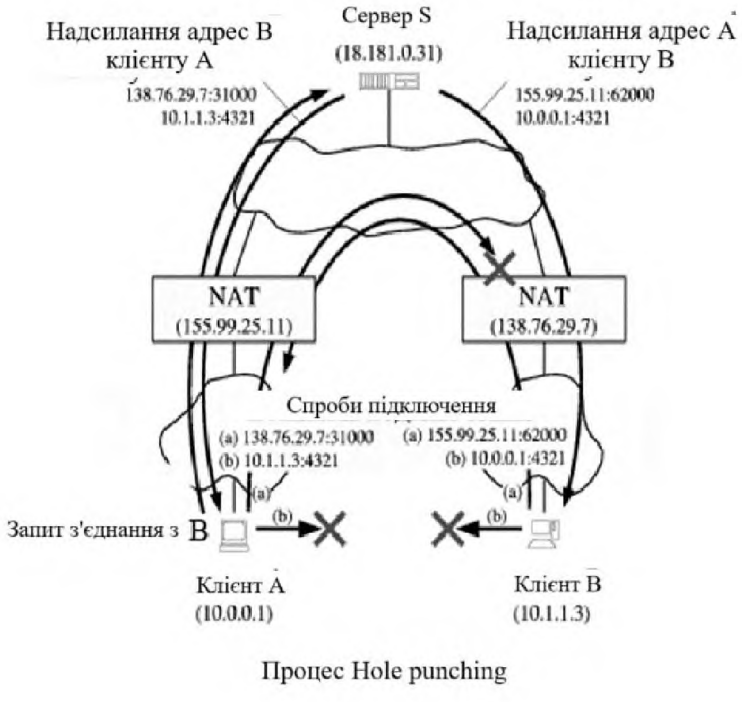

Рисунок 2.10 – Схема з'єднань під час процесу hole punching

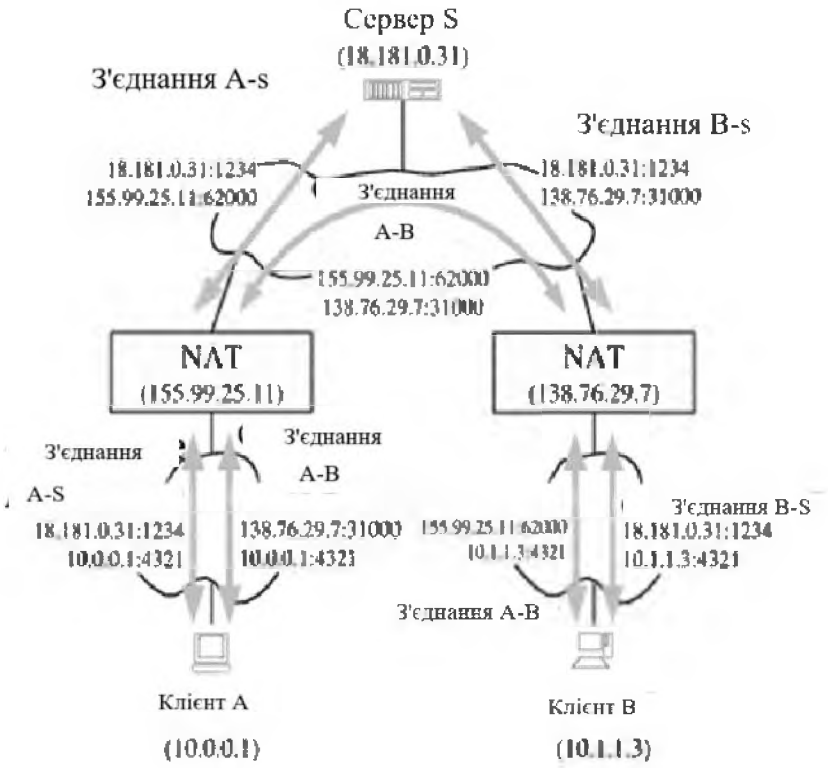

Рисунок 2.11 – Схема з'єднань після процесу hole punching

2.5 Опис роботи клієнтської частини системи

Перейдемо до розгляду деталей роботи клієнтського додатку.

При запуску додаток одночасно запускає процес підключення до сервера та процес ініціалізації в різних потоках. Процес підключення до сервера ми вже розглядали раніше, тому розглянемо відразу процес ініціалізації користувача. Перш за все перевіряється наявність у користувача пари ключів, відкритого та закритого, за допомогою яких буде проводитися асиметричне шифрування. При відсутності цих ключів у користувача, вони генеруються та зберігаються на пристрої. Після успішного з'єднання з сервером та аутентифікації, клієнтський додаток ініціалізує, за потреби, бази даних, які містять список контактів користувача та повідомлення, що не зберігаються на сервері, тобто отримані під час прямого зв'язку з іншим користувачем, створюючи там необхідні таблиці. Зазвичай ця ініціалізація потрібна лише при

першому запуску додатка, однак якщо користувач випадково чи навмисно видалив необхідні додатку дані, вони будуть створені знову.

Після локальної ініціалізації клієнтський додаток надсилає серверу запит на отримання історії повідомлень з його бази даних, одночасно завантажуючи повідомлення та список контактів зі своєї бази даних.

Коли всі необхідні дані отримані, додатку залишається лише реагувати на дії, які вчиняє користувач, та виконувати команди, що приходять від сервера або реагувати на повідомлення інших користувачів.

Найважливіша дія користувача, на яку потрібно реагувати додатку, це, звісно, відправлення повідомлення. Найважливіше при відправленні повідомлення, якщо воно відбувається клієнту напряму, це зберегти його в локальній БД. Якщо повідомлення відправляється у секретній переписці, то його потрібно зашифрувати, процес передачі секретного повідомлення буде розглянуто трохи пізніше.

Команди, що приходять від сервера, зводяться до відповідей сервера на запити, отримані від користувача. Зазвичай вони повідомляють клієнта про успішне або невдале завершення тієї чи іншої дії, наприклад, аутентифікації. Однак деякі повідомлення від сервера містять дані, необхідні для роботи додатка, розглянемо ці дані детальніше.

Дані про публічну та приватну адресу іншого користувача – приватна та публічна IP-адреси та порти клієнта необхідні для встановлення прямого зв'язку з ним.

Дані, що надходять при додаванні людини до списку контактів – успішне виконання запиту на додавання людини до списку контактів, сервер висилає ідентифікаційний номер цієї людини та його відкритий ключ асиметричного шифрування.

Повідомлення з бази даних сервера – повідомлення може бути двох видів: зашифроване або ні, а також воно може бути отримане напряму від іншого користувача або шляхом пересилки повідомлення з сервера. Якщо повідомлення було отримано від користувача напряму, то його необхідно

зберегти в локальній базі даних, оскільки сервер цього повідомлення у себе зберегти не може. Якщо повідомлення було не зашифроване, то воно просто відображається користувачеві, якщо ж воно було зашифроване, то перед відображенням його необхідно розшифрувати, цей процес розглянемо детальніше далі.

Зашифрований ключ з бази даних для розшифровки історії секретних повідомлень – ключ, необхідний для розшифровки секретного повідомлення.

Зашифрований ключ симетричного шифрування для секретного спілкування з іншим користувачем – ключ, який згенерував користувач, бажаючий вести секретне спілкування з нами.

Зараз поговоримо детальніше про те, що відбувається, коли два користувачі вирішують розпочати секретний чат один з одним: як вони обмінюються секретним ключем без розкриття його серверу, і як відновлюється історія секретних повідомлень.

Коли користувач А починає секретну переписку з користувачем В, йому необхідно вибрати ключ, яким він буде шифрувати відправлені повідомлення. Незважаючи на те, що він знає відкриті ключі всіх користувачів із свого списку контактів, застосовувати криптографію з відкритим ключем не бажано, оскільки це вимагає більших витрат часу, ніж шифрування симетричним ключем, особливо це буде помітно при спробі переслати який-небудь файл. Тому додаток генерує ключ симетричного шифрування. Для того, щоб передати ключ користувачеві В, користувач А шифрує його відкритим ключем користувача В за допомогою алгоритму RSA, також користувач А шифрує цей ключ і своїм відкритим ключем, таким чином у нього є два однакових ключа зашифрованих за допомогою різних відкритих ключів. Саме про цю пару ключів йшла мова при розгляді роботи сервера. Сервер збереже цю пару у себе в базі даних, а також перешле користувачу В ключ, зашифрований його відкритим ключем. Користувач В також за допомогою алгоритму RSA і свого закритого ключа розшифрує його і отримає ключ симетричного шифрування

для секретного спілкування з користувачем А. Схема доставки симетричного ключа представлена на рисунку 2.12.

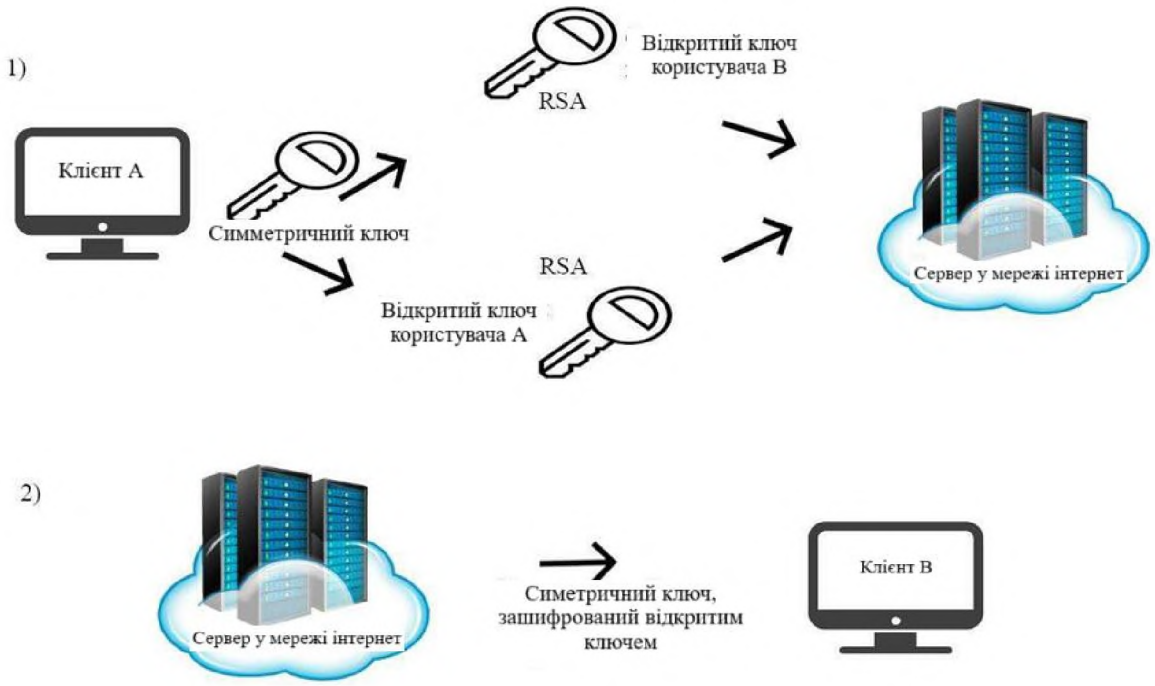

Рисунок 2.12 – Схема встановлення загального симетричного ключа між користувачами

На цьому етапі здійснено обмін ключем таким чином, що ніхто, крім двох кінцевих користувачів, не знає його. Таким чином забезпечується так зване наскрізне шифрування (E2EE – end to end encryption), яке не дозволяє третім особам отримати доступ до приватної переписки [9].

Як уже зазначалося раніше, сервер зберігає всі повідомлення у вихідному вигляді, тому при отриманні доступу до сервера, секретна переписка також не може бути скомпрометована.

Тепер потрібно зрозуміти, як відновити історію зашифрованих повідомлень. Як було розглянуто раніше, коли користувач запитує історію повідомлень, йому також надходять зашифровані ключі, які сервер зберігав у своїй базі даних. Саме для цього користувач на початку секретного спілкування відправляв серверу ключ, який був зашифрований не тільки відкритим ключем

другого користувача, але й своїм власним. Тепер при отриманні даного ключа користувач розшифровує його своїм закритим ключем і, отримуючи зашифровані повідомлення з сервера, може їх розшифрувати.

Загалом робота клієнтського додатка влаштована так, як показано на блок-схемі, представленій на рисунку 2.13.

Тепер розглянемо, які дані зберігає користувач на своєму пристрої. Користувацькі бази даних влаштовані набагато простіше, ніж бази даних сервера, щоб не витрачати багато дискового простору.

Усі повідомлення в базі даних користувача зберігаються у відкритому вигляді, зберігання їх у зашифрованому вигляді не має сенсу. Всі повідомлення зберігаються у таблицях з назвами на кшталт message_history_id або secret_message_history_id, де id – ідентифікаційний номер другого користувача.

Дані таблиці містять однакові поля:

— message_received – поле, що зберігає інформацію про те, чи було дане повідомлення отримано або відправлено;

— message – саме повідомлення;

— mes_type – тип повідомлення, файл або текст.

Приклад повідомлення з таблиці показаний на рисунку 2.15.

База даних, що містить список контактів користувача, містить єдину таблицю friend_list і зберігає в ній наступну інформацію:

— id – ідентифікаційний номер користувача;

— login – логін користувача;

— public_key – відкритий ключ користувача.

Приклад запису з таблиці friend_list показаний рисунку 2.16.

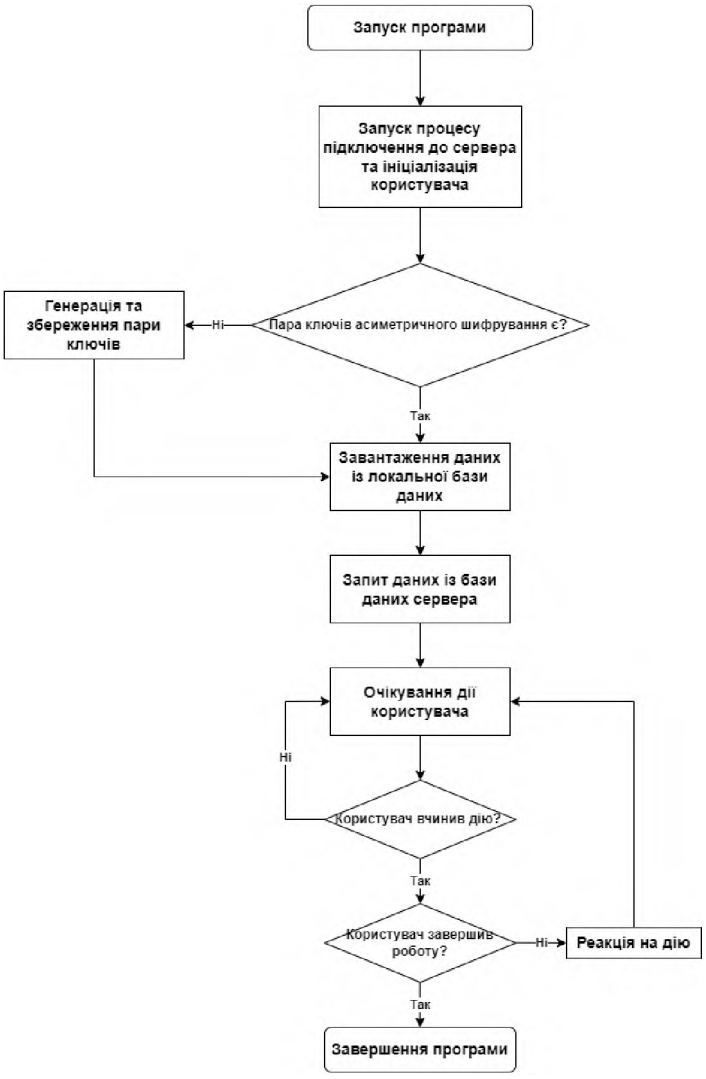

Рисунок 2.14 – Блок-схема роботи клієнтської програми

message_received	message	mes_type
0	104,106	1

Рисунок 2.15 – приклад запису базі даних клієнтських повідомлень

id	login	public_key
1	user1	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,45,45,45,45,10,77,73,73,66,73,106,65,78,66,103,107,113,104,107,105,71,57,119,48,6
5	admin	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,45,45,45,45,10,77,73,73,66,73,106,65,78,66,103,107,113,104,107,105,71,57,119,48,6

Рисунок 2.16 – Приклад запису у базі даних списків контактів користувача

2.6 Опис роботи програми

Ця система складається з кількох модулів, написаних на мові програмування Python, переважно в стилі ООП. Наведемо короткий опис використовуваних модулів для кращого розуміння роботи системи:

— client.py – містить основний функціонал клієнтського додатку;

— client_database.py – клієнтська частина бази даних. У ній зберігається список контактів користувача, а також ті повідомлення, які не зберігаються на сервері;

— common_functions_and_data_structures.py – загальні функції та структури даних, які використовуються як у клієнтській, так і в серверній частинах системи;

— constants.py – список констант;

— database.py – містить основний клас бази даних, від якого успадковані класи, що знаходяться у модулях client_database.py і server_database.py;

— gui2.py – інтерфейс клієнтського додатку;

— main.py – робота клієнтського додатку починається у цьому модулі. Він пов'язує між собою інтерфейс клієнтського додатку з модуля gui2.py і його функціонал з модуля client.py;

— server.py – містить функціонал, необхідний для роботи серверної частини системи;

— server_database.py – серверна частина бази даних. У ній зберігаються всі повідомлення, що проходять через сервер, список користувачів, зареєстрованих у системі разом з інформацією про них, а також деяка допоміжна інформація, необхідна для правильної роботи системи.

Організація модулів у програмі, залежності між ними та основні класи, які вони містять, відображені на малюнках нижче для клієнтського додатку на рисунку 2.1 і для серверного на рисунку 2.2. Модулі зображені прямокутниками, а стрілками зображено відношення включення між ними,

тобто якщо модуль А включений у модуль Б, то стрілка виходить з модуля А і входить у модуль Б.

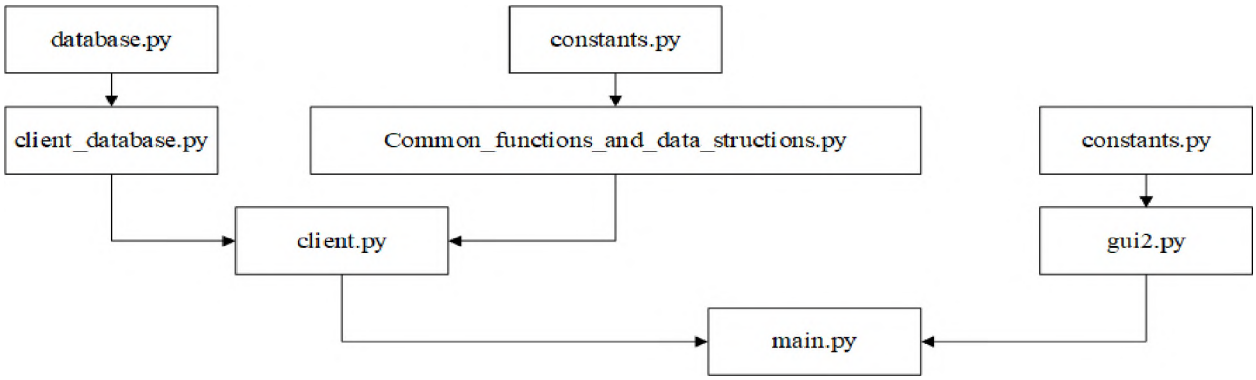

Рисунок 2.1 – Схема організації модулів у клієнтському додатку

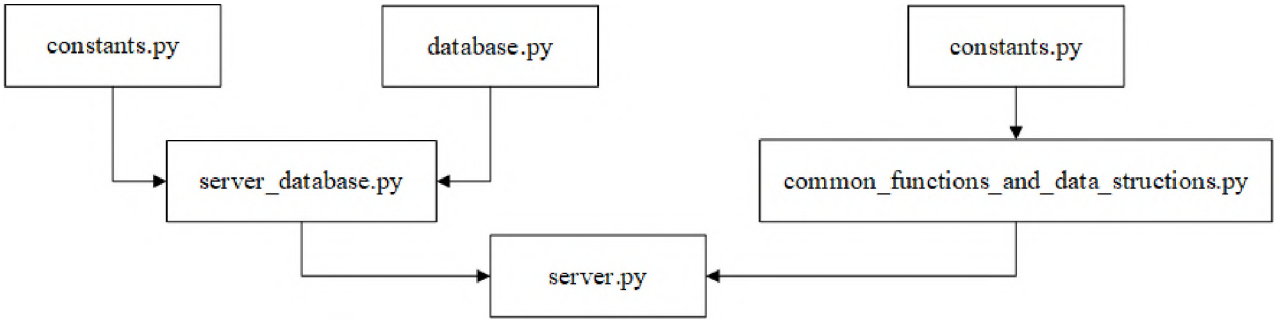

Рисунок 2.2 – Схема організації модулів у серверному додатку

При запуску додатку користувач побачить вікно аутентифікації, як показано на рисунку 2.3. Для продовження роботи необхідно ввести свої дані, а потім увійти або зареєструватися, натиснувши відповідну клавішу.

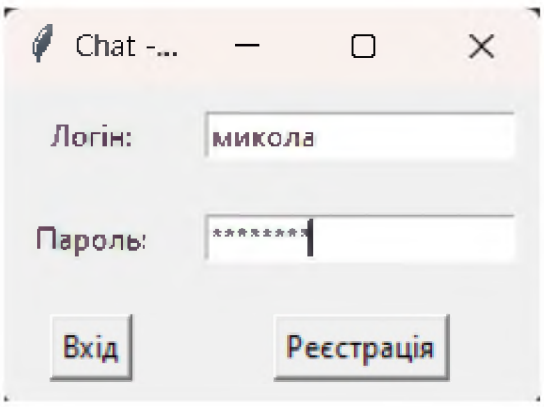

Рисунок 2.3 – Вікно автентифікації

При вдалій реєстрації користувач отримує відповідне повідомлення як показано на рисунку 2.4.

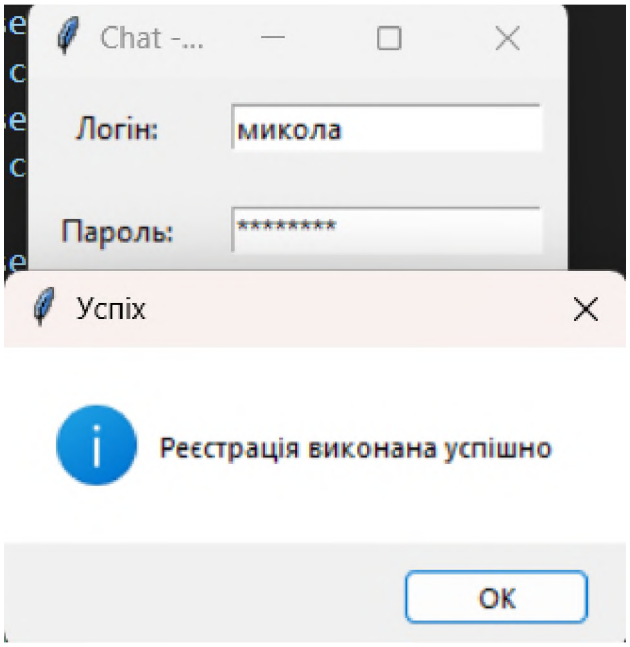

Рисунок 2.4 – Вікно успішної реєстрації

При спробі ввійти з неправильним логіном або паролем користувач отримає попередження, як показано на наведеному нижче рисунку 2.5.

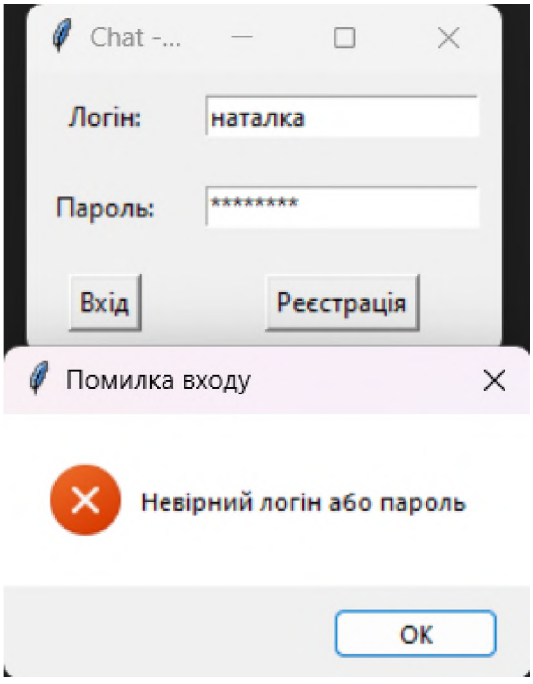

Рисунок 2.5 – Вікно помилки входу

Коли користувач не заповнив всі необхідні поля з’являється відповідне вікно як показано на рисунку 2.6.

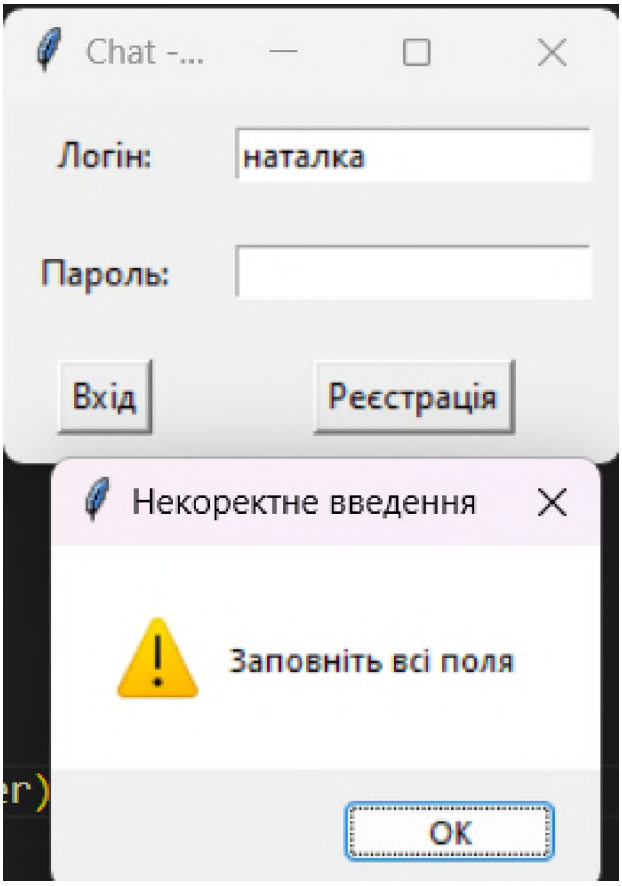

Рисунок 2.6 – Вікно повідомлення про незаповнені поля

Після успішного проходження аутентифікації користувачу відкриється основне вікно клієнтського додатку, показане на рисунку 2.7, в якому відображається список його контактів. Також в головному вікні є можливість додати нового користувача до цього списку або вийти з акаунту.

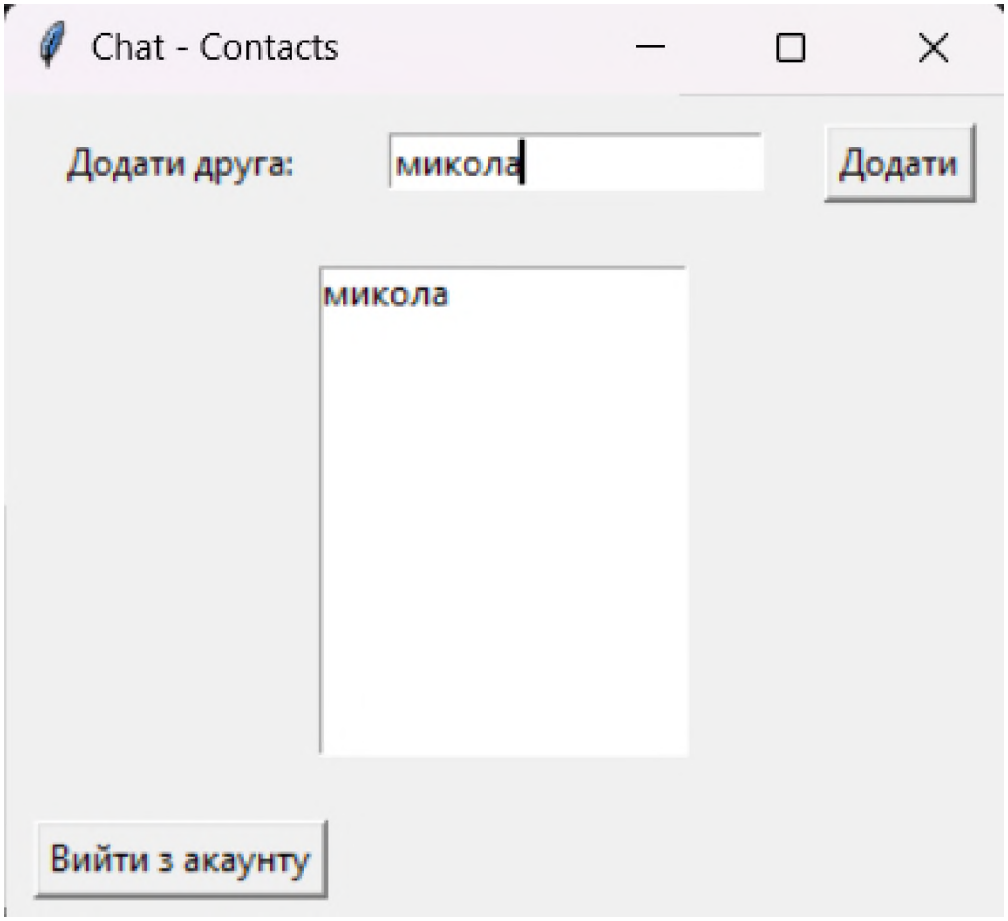

Рисунок 2.7 – Вікно списку контактів

При виборі користувача зі списку, що відображається, відкривається вікно чату з ним, показане на рисунку 2.8. У цьому вікні можна вибрати один з раніше перелічених способів передачі повідомлень, натиснувши відповідну кнопку.

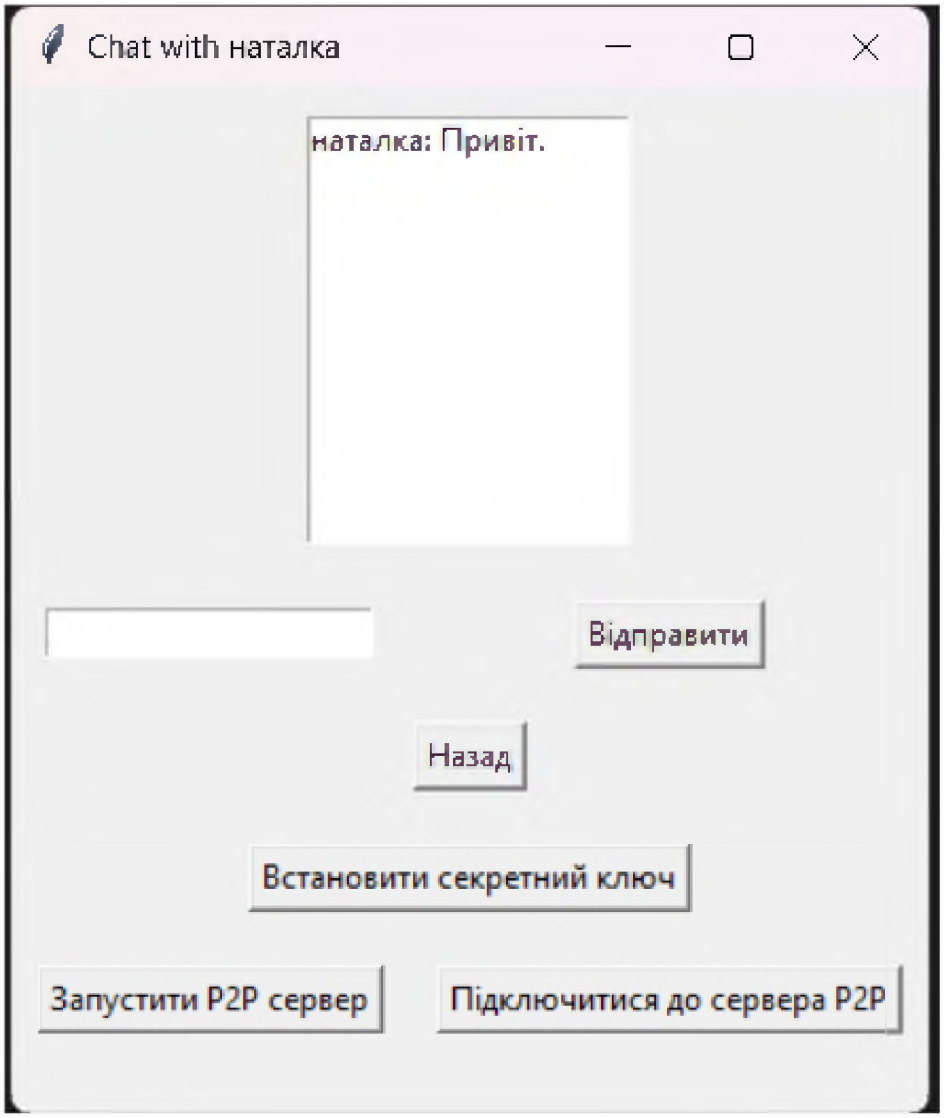

Рисунок 2.8 – Вікно чату

2.7 Висновки до розділу

У цьому розділі розглянуто архітектуру системи обміну повідомленнями в мережі Інтернет, включаючи використані інструменти, можливості реалізованої системи, вибір алгоритмів та способи шифрування, а також опис роботи серверної та клієнтської частин системи.

Для реалізації системи було використано широкий спектр інструментів, таких як мова програмування Python, модулі tkinter для графічного інтерфейсу, SQLite для зберігання даних та бібліотека PyCryptodome для криптографічних функцій та інші Використання цих інструментів забезпечує

кроссплатформеність системи, дозволяючи її використовувати на різних операційних системах, таких як Windows, macOS та Linux.

Реалізована система надає користувачам можливість обмінюватися повідомленнями з використанням різних методів передачі даних, включаючи шифрування повідомлень для забезпечення їхньої конфіденційності та цілісності. Користувачі можуть вибирати між різними способами передачі повідомлень, залежно від їхніх потреб у безпеці та швидкості.

Вибір алгоритмів шифрування базується на ефективності та безпеці. Для симетричного шифрування було обрано алгоритм AES, який забезпечує високу швидкість та безпеку передачі даних. Для асиметричного шифрування використовується алгоритм RSA, що дозволяє надійно обмінюватися ключами. Комбінація цих алгоритмів забезпечує оптимальний баланс між безпекою та продуктивністю системи.

Опис роботи серверної частини системи включає процеси обробки клієнтських запитів, зберігання даних та забезпечення прямого з'єднання між клієнтами. Серверна частина системи відповідає за аутентифікацію користувачів, зберігання історії повідомлень та управління ключами шифрування. Зберігання даних на сервері організовано таким чином, щоб забезпечити їхню цілісність та конфіденційність. Пряме з'єднання клієнтів дозволяє обходити сервер, забезпечуючи більш швидку та захищену передачу даних.

Опис роботи клієнтської частини системи включає процеси ініціалізації, взаємодії з сервером, обробки повідомлень та управління ключами шифрування. Клієнтська частина системи дозволяє користувачам реєструватися, автентифікуватися, додавати контакти та обмінюватися повідомленнями у захищеному режимі. Використання асиметричного та симетричного шифрування забезпечує високий рівень безпеки при передачі даних між клієнтами.

Загалом, розглянута архітектура системи обміну повідомленнями в мережі Інтернет забезпечує високу безпеку та ефективність передачі даних.

Впровадження використаних інструментів, алгоритмів шифрування та методів зберігання даних дозволяє створити надійну та гнучку систему, яка відповідає сучасним вимогам інформаційної безпеки та забезпечує конфіденційність і цілісність переданих даних.

3. ЕКОНОМІЧНИЙ РОЗДІЛ

У наступних підрозділах було досліджено економічну доцільність розробки та впровадження системи обміну повідомленнями з використанням шифрування для корпоративних клієнтів з метою забезпечення безпеки. Основними аспектами економічного аналізу є розрахунок капітальних (фіксованих) витрат, поточних (експлуатаційних) витрат, оцінка загального збитку від можливих атак на підприємство та оцінка загального економічного ефекту від впровадження рекомендацій.

3.2 Розрахунок капітальних (фіксованих) витрат

Капітальні витрати включають витрати на розробку програмного забезпечення показано в таблиці 3.1, закупівлю необхідного обладнання та інструментів, а також витрати на тестування та налагодження системи. Ці витрати є одноразовими та несуть основне навантаження на початковому етапі впровадження системи.

Витрати на розробку програмного забезпечення включають:

— заробітна плата розробників;
— закупівля необхідного програмного забезпечення та інструментів;
— тестування та налагодження системи.

Таблиця 3.1 – Витрати на розробку програмного забезпечення

Категорія витрат	**Сума (USD)**
Заробітна плата розробників (3 розробники, 6 місяців)	90,000
Програмне забезпечення та інструменти	5,000
Тестування та налагодження	10,000
Загалом	**105,000**

Витрати на закупівлю обладнання показані у таблиці 3.2 та включають серверне та мережеве обладнання для забезпечення стабільної роботи мережі.

Таблиця 3.2 – Витрати на обладнання

Категорія витрат	**Сума (USD)**
Серверне обладнання	20,000
Мережеве обладнання	10,000
Загалом	**30,000**

Загальні капітальні витрати відображені у таблиці 3.3 та включають витрати на розробку програмного забезпечення та закупівлю обладнання.

Таблиця 3.3 – Загальні капітальні витрати

Категорія витрат	**Сума (USD)**
Розробка програмного забезпечення	105,000
Закупівля обладнання	30,000
Загалом	**135,000**

Виходячи із представлених даних, загальні капітальні витрати на розробку та впровадження системи безпеки для корпоративних клієнтів, призначеного для безпечного обміну повідомленнями з використанням асиметричного (RSA) та симетричного шифрування становлять 135,000 доларів США. Ці витрати є необхідними для забезпечення високої якості та безпеки системи, що в кінцевому підсумку дозволить знизити ризики та збитки від можливих атак на підприємство, що можуть вплинути на загальні фінансові та репутаційні збитки.

3.3 Розрахунок поточних (експлуатаційних) витрат

Поточні витрати включають витрати на підтримку та обслуговування системи після її впровадження. Ці витрати є постійними та включають заробітну плату персоналу, витрати на обслуговування обладнання, а також витрати на енергію та охолодження.

Для забезпечення стабільної роботи системи необхідно утримувати персонал, що включає адміністраторів системи та технічну підтримку. Заробітна плата персоналу є одним із основних компонентів поточних витрат наведено в таблиці 3.4.

Таблиця 3.4 – Заробітна плата персоналу

Посада	**Кількість працівників**	**Заробітна плата (USD/місяць)**	**Загальні витрати (USD/місяць)**
Адміністратор системи	2	2,000	4,000
Технічна підтримка	2	1,500	3,000
Загалом	**4**		**7,000**

Регулярне обслуговування обладнання є важливим для забезпечення його безперебійної роботи. Це включає в себе оновлення програмного забезпечення, заміну зношених компонентів та загальне технічне обслуговування. Витрати на обслуговування обладнання наведено у таблиці 3.5.

Таблиця 3.5 – Витрати на обслуговування обладнання

Категорія витрат	**Сума (USD/місяць)**
Оновлення програмного забезпечення	1,000
Технічне обслуговування	1,000
Загалом	**2,000**

Серверне та мережеве обладнання споживає значну кількість електроенергії, а також потребує ефективних систем охолодження для забезпечення стабільної роботи, як показано у таблиці 3.6.

Таблиця 3.6 – Витрати на енергію та охолодження

Категорія витрат	**Сума (USD/місяць)**
Електроенергія	1,500
Охолодження	500
Загалом	**2,000**

Загальні поточні витрати (таблиця 3.7) включають заробітну плату персоналу, витрати на обслуговування обладнання, а також витрати на енергію та охолодження.

Таблиця 3.7 – Загальні поточні витрати

Категорія витрат	**Сума (USD/місяць)**
Заробітна плата персоналу	7,000
Обслуговування обладнання	2,000
Енергія та охолодження	2,000
Загалом	**11,000**

Таким чином, загальні поточні (експлуатаційні) витрати на підтримку та обслуговування системи обміну повідомленнями з використанням шифрування становлять 11,000 доларів США на місяць. Ці витрати є необхідними для забезпечення безперебійної та безпечної роботи системи, що дозволить підтримувати високий рівень захищеності даних корпоративних клієнтів.

3.4 Розрахунок загального збитку від атаки на підприємство

Загальний збиток від атаки на підприємство включає прямі та непрямі витрати, що виникають внаслідок компрометації інформаційних систем. До таких збитків належать втрата конфіденційної інформації, фінансові втрати, витрати на відновлення систем та репутаційні втрати. У цьому розділі проведемо розрахунок загального збитку, що може бути завданий підприємству в результаті успішної атаки. Прямі збитки включають втрати (таблиця 3.8), які можна безпосередньо пов'язати з атакою, такі як втрата конфіденційної інформації та фінансові втрати. Втрата конфіденційної інформації може призвести до серйозних наслідків, таких як витік комерційних таємниць або особистих даних клієнтів. Приклад розрахунку: $50,000. Фінансові витрати, пов'язані з прямим вилученням коштів або втрати через неможливість здійснення операцій під час атаки. Приклад розрахунку: $30,000.

Таблиця 3.8 – Прямі збитки

Категорія збитків	**Сума (USD)**
Втрата конфіденційної інформації	50,000
Фінансові втрати	30,000
Загалом	**80,000**

Непрямі збитки включають втрати (таблиця 3.9), які виникають опосередковано внаслідок атаки, такі як зниження репутації компанії, витрати на відновлення системи та втрати клієнтів.

Зниження репутації компанії та втрата довіри клієнтів та партнерів, що може призвести до зменшення прибутків. Приклад розрахунку: $40,000. Витрати на відновлення пошкоджених систем та відновлення втрачених даних. Приклад розрахунку: $20,000. Втрата клієнтів, які залишають компанію через недовіру до безпеки їх даних. Приклад розрахунку: $30,000.

Таблиця 3.9 – Непрямі збитки

Категорія збитків	**Сума (USD)**
Зниження репутації компанії	40,000
Витрати на відновлення системи	20,000
Втрата клієнтів та доходів	30,000
Загалом	**90,000**

Загальні збитки від атаки включають суму прямих та непрямих збитків (таблиця 3.10).

Таблиця 3.10 – Загальні збитки від атаки

Категорія збитків	**Сума (USD)**
Прямі збитки	80,000
Непрямі збитки	90,000
Загалом	**170,000**

Таким чином, загальні збитки від атаки на підприємство можуть становити 170,000 доларів США. Ця сума включає як прямі фінансові втрати, так і непрямі збитки, пов'язані з репутаційними втратами та відновленням системи. Враховуючи ці потенційні збитки, впровадження системи безпечного обміну повідомленнями з використанням асиметричного (RSA) та симетричного шифрування стає економічно доцільним заходом для зменшення ризиків.

3.5 Загальний ефект від впровадження рекомендацій

Впровадження системи безпечного обміну для корпоративних клієнтів має на меті підвищення рівня безпеки та зменшення ризиків. У цьому розділі було розглянуто економічний ефект від впровадження рекомендацій, враховуючи зменшення ймовірності успішних атак, підвищення репутації компанії, а також економію на інших заходах безпеки.

Одним із головних ефектів впровадження системи є зменшення ймовірності успішних атак, що призводить до зниження витрат на відновлення після атак та збереження конфіденційної інформації.

Зменшення витрат на відновлення після атак, впровадження системи знижує ризик компрометації даних, що зменшує необхідність витрат на відновлення системи після атак. Приклад розрахунку: $50,000/рік.

Збереження конфіденційної інформації, завдяки надійному шифруванню даних, зменшується ймовірність витоку конфіденційної інформації, що дозволяє зберегти фінансові ресурси. Приклад розрахунку: $30,000/рік.

Таблиця 3.11 – Ефект від зменшення ймовірності успішних атак

Категорія витрат	**Сума (USD/рік)**
Зменшення витрат на відновлення	50,000
Збереження конфіденційної інформації	30,000
Загалом	**80,000**

Високий рівень безпеки даних сприяє підвищенню репутації компанії серед клієнтів та партнерів, що позитивно впливає на фінансові показники.

Підвищення рівня довіри клієнтів до безпеки їх даних стимулює зростання клієнтської бази та доходів компанії. Приклад розрахунку: $30,000/рік.

Забезпечення високого рівня безпеки даних робить компанію більш привабливою для нових клієнтів та партнерів. Приклад розрахунку: $20,000/рік.

Таблиця 3.12 – Ефект від підвищення репутації компанії

Категорія витрат	**Сума (USD/рік)**
Збільшення довіри клієнтів	30,000
Підвищення конкурентоспроможності	20,000
Загалом	**50,000**

Впровадження системи знижує необхідність використання інших, менш ефективних засобів безпеки, що призводить до економії коштів.

Зменшення витрат на інші засоби безпеки:

— впровадження ефективної системи шифрування дозволяє скоротити витрати на інші заходи безпеки, такі як додаткове апаратне забезпечення або програмне забезпечення для захисту даних. Приклад розрахунку: $20,000/рік.

Таблиця 3.13 – Ефект від економії на інших заходах безпеки

Категорія витрат	**Сума (USD/рік)**
Зменшення витрат на інші засоби безпеки	20,000
Загалом	**20,000**

Загальний економічний ефект від впровадження системи включає зменшення витрат на відновлення після атак, збереження конфіденційної інформації, підвищення репутації компанії та економію на інших заходах безпеки.

Таблиця 3.14 – Загальний економічний ефект

Категорія витрат	**Сума (USD/рік)**
Зменшення витрат на відновлення після атак	50,000
Збереження конфіденційної інформації	30,000
Збільшення довіри клієнтів	30,000
Підвищення конкурентоспроможності	20,000
Економія на інших засобах безпеки	20,000
Загалом	**150,000**

Роблячи висновок, загальний економічний ефект від впровадження системи обміну повідомленнями з використанням асиметричного та симетричного шифрування становить 150,000 доларів США на рік. Це підтверджує економічну доцільність впровадження системи та демонструє її значний позитивний вплив на фінансові показники та безпеку корпоративних клієнтів.

ВИСНОВКИ

В ході виконання бакалаврської роботи були розглянуті методи передачі даних по різних мережах, а також методи та технології захисту передаваних даних. Для кращого розуміння використовуваних методів безпеки були розібрані основні принципи сучасної криптографії, розглянуті алгоритми як симетричного, так і асиметричного шифрування, на яких вони побудовані.

Було реалізовано систему мовою програмування Python, що складається з двох додатків: клієнтського та серверного. Користувачі цієї системи мають можливість спілкуватися один з одним у мережі Інтернет, причому спілкування може відбуватися в секретному режимі, тобто таким чином, що треті особи не отримають доступ до переписки. Окрім звичайних повідомлень, система може передавати файли через мережу. Всі повідомлення зберігаються в базі даних, і навіть у разі отримання доступу до неї третьою особою, жодна секретна інформація не буде скомпрометована, оскільки вона зберігається в зашифрованому вигляді.

Звісно, ця система не є ідеальною і має свої недоліки в плані зручності використання, проте в плані захисту даних вона не поступається системам, над якими працюють цілі команди розробників у великих компаніях.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шифрування та як воно працює. URL: https://www.kingston.com/ua/blog/data-security/what-is-encryption (дата звернення: 05.05.2024)

2. Симетричне шифрування. URL: https://academy.binance.com/uk/articles/symmetric-vs-asymmetric-encryption (дата звернення: 07.05.2024)

3. Алгоритм Діффі-Геллмана. URL: https://dudkabm12.wordpress.com/2014/10/15/%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC-%D0%B4%D0%B5%D1%84%D1%84%D1%96-%D1%85%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD%D0%B0/ (дата звернення: 08.05.2024)

4. Алгоритм шифрування RSA. URL: https://dou.ua/forums/topic/43026/ (дата звернення: 10.05.2024)

5. Загальні відомості про цифрові підписи. URL: https://support.microsoft.com/uk-ua/office/%D0%B7%D0%B0%D0%B3%D0%B0%D0%BB%D1%8C%D0%BD%D1%96-%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BE%D1%81%D1%82%D1%96-%D0%BF%D1%80%D0%BE-%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D1%96-%D0%BF%D1%96%D0%B4%D0%BF%D0%B8%D1%81%D0%B8-d2f92222-abb1-486b-bc07-884ecac99c59 (дата звернення: 13.05.2024)

6. SSL/TLS. URL: https://vps.ua/blog/ukr/ssl-tls-protocols-details/ (дата звернення: 15.05.2024)

7. Документація Python. URL: https://docs.python.org/3/ (дата звернення: 20.05.2024)

8. Трансляція мережевих адрес (NAT). URL: https://hyperhost.ua/info/uk/shcho-take-nat-dlya-chogo-vikoristovuyu-daniy-standart (дата звернення: 25.05.2024)

9. Наскрізне шифрування (E2EE). URL: https://academy.binance.com/uk/articles/what-is-end-to-end-encryption-e2ee (дата звернення: 30.05.2024)

ДОДАТОК А. Відомість матеріалів кваліфікаційної роботи

№	Формат	Найменування	Кількість листів	Примітка
1	А4	Реферат	2	
2	А4	Список умовних скорочень	1	
3	А4	Зміст	2	
4	А4	Вступ	2	
5	А4	1 Розділ	17	
6	А4	2 Розділ	31	
7	А4	3 Розділ	9	
8	А4	Висновки	1	
9	А4	Перелік посилань	2	
10	А4	Додаток А	1	
11	А4	Додаток Б	1	
12	А4	Додаток В	1	
13	А4	Додаток Г	2	

ДОДАТОК Б. Перелік документів на оптичному носії

Пояснювальна записка.docx

Презентація.pptx

ДОДАТОК В. Відгуки керівників розділів

Відгук керівника економічного розділу:

Економічний розділ виконаний відповідно до вимог, які ставляться до кваліфікаційних робіт, та заслуговує на оцінку ____ б. («_____»).

Керівник розділу _____ Дар'я ПІЛОВА

(підпис) (ім'я, прізвище)

ДОДАТОК Г. ВІДГУК

на кваліфікаційну роботу бакалавра на тему: Аналіз методів симетричного шифрування для захисту корпоративних даних

студента групи 125-20-1
Олексюк Валерії Олександрівни

Пояснювальна записка складається зі вступу, трьох розділів і висновків, розташованих на 75 сторінках.

Мета роботи є актуальною, оскільки вона спрямована на розробку системи обміну повідомленнями з використанням асиметричного і симетричного шифрування для забезпечення безпеки передачі даних.

При виконанні роботи авторка продемонструвала добрий рівень теоретичних знань і практичних навичок. На основі аналізу сучасних методів шифрування даних, організаційного забезпечення обміну повідомленнями в мережі Інтернет, а також визначення актуальності впровадження наскрізного шифрування в ній сформульовано задачі, вирішенню яких присвячений спеціальний розділ. У ньому було запропоновано архітектуру системи обміну повідомленнями в мережі Інтернет, а також надано рекомендації щодо реалізації безпечного обміну повідомленнями.

Практична цінність роботи полягає у тому, що запропоновані рішення можуть бути використані для підвищення безпеки процесу обміну повідомленнями в мережі Інтернет.

До недоліків роботи слід віднести недостатню проробку окремих питань.

Рівень запозичень у кваліфікаційній роботі не перевищує вимог «Положення про систему виявлення та запобігання плагіату».

В цілому робота задовольняє усім вимогам, а її авторка Олексюк Валерія Олександрівна заслуговує на оцінку « » та присвоєння кваліфікації «Бакалавр з кібербезпеки» за спеціальністю 125 Кібербезпека.

Керівник роботи,

к.т.н., доцент **О.В. Герасіна**