Міністерство освіти і науки України

Криворізький національний університет

Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня віщої освіти магістра

зі спеціальності 121 — Інженерія програмного забезпечення

На тему: «Дослідження та розробка систем захисту даних з використанням пост-квантової криптографії»

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент групи ІПЗ-23-1м

Мухаммад Даніш Хан

Керівник
кваліфікаційної роботи _____ О. В. Шамрай

Завідувач кафедри _____ А. М. Стрюк

Кривий Ріг

2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 — Інженерія програмного забезпечення

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи

1. Тема: «Дослідження та розробка систем захисту даних з використанням пост-квантової криптографії» затверджено наказом по КНУ №111 від «1» березня 2024 р.

2. Термін подання студентом закінченої роботи: «30» листопада 2024 р.

3. Вихідні дані по роботі: розроблена система повинна надавати можливість користувачам безпечно обмінюватися даними захищеними методами пост-квантової криптографії.

4. Зміст пояснювальної записки (перелік питань, що треба розробити): проаналізувати сучасні методи криптографії, їх реалізації та ефективність в контексті майбутніх можливих квантових атак.

5. Перелік ілюстративного матеріалу: зображення аналогів, функціональна схема, блок-схеми алгоритмів програми, зображення інтерфейсу програми, ER-діаграма бази даних.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Ознайомлення з джерелами за темою дослідження	01.09.2024 - 14.09.2024
2	Формування цілей та завдання кваліфікаційної роботи	15.09.2024 - 19.09.2024
3	Дослідження сфер застосування криптографічних методів	20.09.2024 - 26.10.2024
	Реалізація системи з урахуванням потенційної небезпеки з боку квантових комп’ютерів.	27.10.2024 - 04.11.2024
5	Оформлення пояснювальної записки	05.11.2024 - 01.12.2024

Дата видачі завдання «01» вересня 2024 р.

Студент _____ / Д. Х. Мухаммад /

Керівник роботи _____ / О. В. Шамрай /

РЕФЕРАТ

Ключові слова: КРИПТОГРАФІЯ, КВАНТОВИЙ, ПОСТ-КВАНТОВИЙ, КВАНТОВИЙ КОМП'ЮТЕР, СИМЕТРИЧНИЙ, АСИМЕТРИЧНИЙ, КЛЮЧ, КРИПТОСТІЙКІСТЬ, RSA, TLS, ECDSA, AES, ECDHE, KYBER, СЕСІЙНИЙ КЛЮЧ, ЛЮДИНА ПОСЕРЕДИНІ.

Пояснювальна записка: 96 с., 5 рис, 2 дод., 15 джерел, 13 формул.

Мета кваліфікаційної роботи − дослідження та розробка системи захисту даних із використанням методів постквантової криптографії.

Об'єкт проектування – система безпеки даних, що забезпечує захист інформації за допомогою постквантових криптографічних алгоритмів.

У теоретичній частині кваліфікаційної роботи проведено аналіз сучасних методів криптографії, включаючи симетричні, асиметричні та хешувальні алгоритми. Досліджено вразливість традиційних криптографічних методів до атак квантових обчислень. Особлива увага приділена аналізу постквантових криптографічних алгоритмів, таких як Kyber та Dilithium, з точки зору їхньої стійкості та ефективності. Сформульовані актуальність та завдання дослідження.

У практичній частині кваліфікаційної роботи розроблено архітектуру системи, що інтегрує постквантові алгоритми для захисту даних у веб-додатку. Запропоновано та впроваджено використання алгоритму Kyber для обміну ключами у фінансових транзакціях з підтримкою Perfect Forward Secrecy.

У спеціальній частині розроблено програмний комплекс, що включає модулі перевірки доступу та захисту сесій користувачів. Проведено тестування ефективності розроблених рішень, яке показало відповідність системи вимогам безпеки та продуктивності. Зроблено висновки, що впровадження постквантових методів у сфери обміну та збереження інформації забезпечує високий рівень захисту в умовах можливого використання квантових обчислень.

ABSTRACT

Keywords: CRYPTOGRAPHY, QUANTUM, POST-QUANTUM, QUANTUM COMPUTER, SYMMETRIC, ASYMMETRIC, KEY, CRYPTOSTENSITY, RSA, TLS, ECDSA, AES, ECDHE, KYBER, SESSION KEY, MAN IN THE MIDDLE.

Thesis in 96 p., 5 pics, 2 pp., 15 sources, 13 formulas.

The aim of the qualification work is to research and develop a data protection system using post-quantum cryptographic methods.

The design object is a data security system that ensures information protection using post-quantum cryptographic algorithms.

The theoretical part of the qualification work includes an analysis of modern cryptographic methods, including symmetric, asymmetric, and hash algorithms. The vulnerabilities of traditional cryptographic methods to quantum computing attacks were investigated. Special attention was paid to analyzing post-quantum cryptographic algorithms such as Kyber and Dilithium in terms of their resilience and efficiency. The relevance and objectives of the study were formulated.

In the practical part of the qualification work, a system architecture was developed that integrates post-quantum algorithms for data protection in a web application. The use of the Kyber algorithm for key exchange in financial transactions with support for Perfect Forward Secrecy was proposed and implemented.

In the specialized section, a software complex was developed, including modules for access control and user session protection. The effectiveness of the developed solutions was tested, demonstrating the system’s compliance with security and performance requirements. It was concluded that the implementation of post-quantum methods in information exchange and storage domains ensures a high level of protection under potential quantum computing threats.

1. ЗМІСТ

ВСТУП

У сучасному світі, де інформація стала ключовим ресурсом, забезпечення її захисту є однією з найважливіших задач інформаційної безпеки. Зростання обчислювальних потужностей, розвиток штучного інтелекту та впровадження квантових обчислень створюють нові виклики для криптографії — науки, яка стоїть на сторожі конфіденційності, цілісності та автентичності даних. Особливе занепокоєння викликає перспектива створення квантових комп’ютерів, здатних ефективно вирішувати математичні задачі, на яких базуються сучасні криптографічні алгоритми.

Існуючі алгоритми, такі як RSA, ECC (еліптичні криві) та навіть симетричні методи, наприклад AES із меншими ключами, можуть стати вразливими до атак квантових комп’ютерів. Наприклад, алгоритм Шора дозволяє ефективно зламувати асиметричні алгоритми шляхом розкладання великих чисел на прості множники чи обчислення дискретного логарифму. Подібним чином алгоритм Гровера може скоротити час перебору ключів для симетричних шифрів, що знижує ефективну криптостійкість таких методів. У результаті виникає необхідність створення нових підходів до захисту даних, які зможуть протистояти квантовим обчисленням.

У зв’язку з цим пост-квантова криптографія стає надзвичайно актуальним напрямком досліджень. Вона базується на задачах, які залишаються складними навіть для квантових комп’ютерів, таких як решіткові задачі, декодування надлишкових кодів чи ізогенії еліптичних кривих. Сучасні наукові дослідження у цій галузі спрямовані на розробку алгоритмів, які забезпечать високу стійкість до квантових атак і при цьому залишатимуться ефективними для реалізації у реальних умовах.

Метою даної дипломної роботи є дослідження сучасних криптографічних методів, їх реалізацій у популярних програмних системах, а також порівняння існуючих пост-квантових підходів. Робота передбачає вибір найбільш перспективного пост-квантового алгоритму для захисту даних, його реалізацію, а також створення веб-додатку для онлайн-банкінгу з

використанням цього алгоритму. Це дозволить не лише поглибити розуміння сучасних проблем криптографії, але й запропонувати практичне рішення, яке відповідатиме майбутнім викликам у сфері інформаційної безпеки.

Таким чином, ця робота спрямована на внесення вкладу у розвиток систем захисту даних, які залишатимуться актуальними навіть в епоху квантових обчислень.

1. СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Актуальність теми кваліфікаційної роботи.

Криптографія, наука про шифрування та захист інформації, має глибоке історичне коріння. Її розвиток простежується з давніх часів до сучасності, і вона завжди грала ключову роль у захисті важливої інформації.

Важливим моментом у розвитку криптографії стало її застосування під час світових воєн. Наприклад, у Першій світовій війні використовувався шифр АДФГВХ, а у Другій світовій війні — знаменитий німецький шифрувальний апарат "Енігма". Розшифровка повідомлень "Енігми" британськими криптографами в Блетчлі Парк, на чолі з Аланом Тьюрінгом, стала одним з ключових моментів війни.

Після Другої світової війни криптографія стала важливою складовою військової та цивільної безпеки. У 1970-х роках було розроблено «Data Encryption Standard» (DES), який став першим широко використовуваним стандартом цифрового шифрування. Проте, через обмеження в довжині ключа в 56 біт, DES став вразливим до "грубої сили" — атак, які використовують величезні обчислювальні потужності для перебору всіх можливих ключів.

У відповідь на слабкості DES у 2001 році було прийнято новий стандарт — Advanced Encryption Standard (AES), який використовує ключі довжиною 128, 192, або 256 біт, що надає значно кращий рівень безпеки.

Останніми роками з'явилась нова загроза для сучасних криптографічних систем - квантові обчислення. Квантові комп'ютери мають потенціал розв'язати деякі математичні задачі набагато швидше, ніж це можливо з використанням класичних комп'ютерів, що може зробити вразливими навіть стійкі на даний момент системи, такі як AES. Це стимулює науковців та розробників шукати нові шляхи захисту даних, такі як пост-квантова криптографія, яка може забезпечити безпеку в епоху квантових технологій.

Квантові комп’ютери — це пристрої, які виконують обчислення, використовуючи закони квантової механіки. На відміну від класичних

комп’ютерів, які працюють з бітами (0 або 1), квантові комп’ютери використовують кубіти, які можуть перебувати у стані 0, 1 або суперпозиції обох. Завдяки цьому квантові комп’ютери теоретично здатні вирішувати певні задачі значно швидше, ніж класичні обчислювальні машини.

Вони використовують закони квантової механіки, мають потенціал для розв’язання задач, які недоступні для класичних обчислювальних систем. Завдяки використанню кубітів, здатних перебувати у стані 0, 1 або їхньої суперпозиції, квантові комп’ютери можуть значно пришвидшити виконання певних обчислень. На сьогодні провідними розробниками є компанії IBM, Google, Rigetti, IonQ та D-Wave. Наприклад, IBM має квантовий процесор із 433 кубітами, а Google у 2019 році оголосила про досягнення “квантової переваги”, виконавши обчислення, які класичний комп’ютер виконував би десятки тисяч років, за кілька хвилин. Проте ці успіхи поки що не означають практичної можливості зламати сучасні криптографічні алгоритми.

Для зламу асиметричних алгоритмів, таких як RSA чи ECDSA, квантовий комп’ютер може використовувати алгоритм Шора. Цей алгоритм здатний розкладати великі числа на множники чи вирішувати задачу дискретного логарифма, що лежить в основі цих протоколів. Однак для зламу RSA-2048 потрібні приблизно 20 мільйонів логічних кубітів та кілька днів обчислень. Сучасні квантові комп’ютери мають лише сотні фізичних кубітів, тоді як створення одного логічного кубіта потребує об’єднання десятків фізичних кубітів для корекції помилок. Це значно ускладнює реалізацію атаки на практиці.

Симетричні алгоритми, такі як AES-128-GCM, також вразливі до квантових обчислень через алгоритм Гровера, який скорочує час перебору ключів з 2^128 до 2^64. Проте навіть 2^64 обчислень — це величезний обсяг роботи, недосяжний для сучасних квантових комп’ютерів. Для таких атак потрібні сотні мільйонів логічних кубітів і значні обчислювальні ресурси, яких наразі не існує.

Сучасні квантові комп’ютери обмежені кількістю кубітів, високим

рівнем шуму, коротким часом збереження стану (декогеренцією) та технічними викликами корекції помилок. Навіть найпотужніші системи сьогодні не здатні зламати сучасні криптографічні протоколи на практиці. Проте технології швидко розвиваються, і експерти оцінюють, що для зламу RSA-2048 знадобиться ще близько 10-20 років досліджень і розробок. Для захисту в довгостроковій перспективі вже зараз впроваджуються пост-квантові алгоритми, такі як Kyber чи Dilithium, які базуються на задачах, не вразливих до квантових обчислень. Сучасна криптографія є безпечною в короткостроковій перспективі, але перехід на пост-квантову криптографію є стратегічно важливим для збереження конфіденційності даних у майбутньому.

1.2 Цілі та завдання кваліфікаційної роботи.

Актуальність даного дослідження з пост-квантової криптографії пояснюється кількома ключовими чинниками, що визначають сучасні тенденції в технологічному розвитку і безпеці даних.

Технологічний прогрес у квантових обчисленнях. Розвиток квантових комп'ютерів вже не тільки теоретичною можливістю, але наближається до практичної реалізації. Ці комп'ютери обіцяють значне зростання обчислювальних потужностей, здатних ефективно вирішувати задачі, які зараз вважаються нерозв'язними для класичних обчислювальних систем. Це стосується, зокрема, алгоритмів шифрування, базованих на факторизації великих чисел та дискретних логарифмах, які є основою для багатьох сучасних криптосистем.

Вразливість існуючих алгоритмів шифрування. Більшість існуючих алгоритмів, таких як RSA та ECC, виявляться абсолютно неефективними проти атак з використанням квантових комп'ютерів. Алгоритм Шора, наприклад, може ефективно розв'язувати такі задачі, що ставить під загрозу конфіденційність і цілісність персональних, корпоративних, фінансових та державних даних.

Необхідність нових стандартів безпеки. Враховуючи можливість

виникнення квантових комп'ютерів, необхідно розробити нові криптографічні стандарти, які можуть протистояти квантовим атакам. Це вимагає великомасштабних досліджень в галузі пост-квантової криптографії, щоб забезпечити плавний перехід від класичних методів до нових, що зможуть забезпечити захист інформації в квантову епоху.

Глобальні наслідки для інформаційної безпеки. Враховуючи глобалізацію та зростаючу взаємозалежність економік, вразливості в криптографічних системах можуть мати далекосяжні наслідки. Злом криптосистем може призвести до втрати важливої інформації, фінансових ресурсів, а також підірвати довіру до цифрових систем у всьому світі.

Соціальна та економічна важливість. У епоху цифрової економіки безпека даних є ключовою для забезпечення стабільності та довіри в електронній комерції, цифрових комунікаціях та національній безпеці. Пост-квантова криптографія надає засоби для забезпечення цієї безпеки в довгостроковій перспективі.

Інноваційність у наукових дослідженнях. Розвиток пост-квантової криптографії сприяє не тільки захисту даних, а й стимулює наукову спільноту до нових відкриттів. Це підштовхує до глибшого розуміння математичних проблем, які лежать в основі криптографічних алгоритмів, і приводить до виникнення нових наукових напрямків і спеціалізацій.

Попередження кіберзлочинності та захист критичної інфраструктури: Квантові комп'ютери можуть значно збільшити ризики пов'язані з кіберзлочинністю, адже з їх допомогою зловмисники можуть швидко розшифровувати захищені дані. Розробка пост-квантових алгоритмів дозволить значно підвищити рівень безпеки урядових і комерційних інформаційних систем, забезпечуючи захист від масштабних кібератак на критично важливі інфраструктури, такі як енергетичні мережі, фінансові інститути та національні безпекові системи.

Міжнародне співробітництво та стандартизація: Розвиток пост-квантової криптографії має важливе значення для міжнародного

співробітництва у сфері кібербезпеки. Адаптація нових стандартів на міжнародному рівні вимагатиме широкомасштабної координації між урядами, промисловістю та академічними кілами, що сприятиме уніфікації підходів до захисту інформації та зміцненню міжнародної безпеки.

Етичні та правові аспекти: Впровадження пост-квантової криптографії також порушує питання про етику у використанні криптографічних технологій та про вплив на права і приватність осіб. Необхідність розробки міжнародних правил, які регулюватимуть використання та розповсюдження пост-квантових технологій, важлива для забезпечення, що інновації слугують загальному благу та не викликають зловживань.

Робота має на меті дослідження та розробку ефективних систем захисту даних з використанням пост-квантової криптографії, які здатні протистояти загрозам, що виникають через стрімке зростання обчислювальних потужностей. Враховуючи, що сучасні криптографічні алгоритми можуть бути зламані з появою квантових комп'ютерів, дослідження має на меті підготувати криптографічні рішення для забезпечення безпеки даних в умовах нової ери квантових обчислень, до якої ми стоїмо на порозі.

Завдання дослідження:

а) аналіз сучасних криптографічних алгоритмів та їх вразливостей у контексті квантових обчислень;

б) вивчення принципів і методів пост-квантової криптографії: дослідження математичних основ алгоритмів, стійких до атак квантових комп'ютерів;

в) оцінка ефективності та безпеки існуючих пост-квантових криптографічних алгоритмів з точки зору продуктивності та захищеності;

г) моделювання та розробка систем захисту даних на основі пост-квантових алгоритмів для різних інформаційних систем;

г) проведення експериментів з впровадження пост-квантової криптографії у реальних умовах та оцінка її впливу на безпеку й продуктивність систем;

д) вивчення потенційних викликів та обмежень впровадження пост-квантової криптографії, таких як масштабованість та інтеграція в існуючу інфраструктуру.

Таким чином, тема дослідження є надзвичайно актуальною і необхідно для подальшого розвитку інформаційної безпеки, адаптації до нових технологічних умов і забезпечення стабільності та безпеки в цифровому світі.

2. ДОСЛІДЖЕННЯ МЕТОДІВ КРИПТОГРАФІЇ

2.1 Дослідження сфер застосування криптографічних методів.

2.1.1 Електронна комерція.

Електронна комерція, або e-commerce — це процес купівлі, продажу або обміну товарами, послугами чи інформацією через Інтернет. Вона охоплює широкий спектр бізнес-діяльності: від інтернет-магазинів і маркетплейсів до платіжних систем і платформ для обміну послугами. Основними моделями електронної комерції є B2B (бізнес для бізнесу), B2C (бізнес для споживача), C2C (споживач для споживача) та G2C (уряд для споживача). Цей підхід до торгівлі став фундаментальною частиною глобальної економіки та має надзвичайний вплив на сучасний спосіб життя.

Електронна комерція в сучасному світі є надзвичайно важливою через її здатність трансформувати традиційну економічну діяльність. Вона забезпечує швидкість і зручність для споживачів, дозволяючи їм купувати товари або замовляти послуги з будь-якої точки світу у будь-який час. Споживачі можуть порівнювати ціни, читати відгуки та знаходити найкращі пропозиції, не залишаючи своїх домівок. Для бізнесу електронна комерція відкриває можливість значно розширити ринки без значних інвестицій у фізичну інфраструктуру. Наприклад, компанії можуть продавати свої товари на глобальному рівні, створюючи онлайн-платформи, які легко масштабується відповідно до попиту. Зменшення витрат на оренду приміщень і оплату персоналу робить цей підхід економічно вигідним для бізнесу будь-якого розміру.

Інновації в технологіях також сприяють розвитку електронної комерції. Використання штучного інтелекту дозволяє автоматизувати процеси, такі як персоналізація рекомендацій, обробка замовлень і підтримка клієнтів через чат-боти. Великі дані допомагають бізнесу краще розуміти потреби споживачів і адаптувати свої пропозиції до їхніх уподобань. Крім того, електронна комерція сприяє фінансовій інклюзії, оскільки надає доступ до

товарів і послуг людям, які не мають можливості здійснювати покупки офлайн, наприклад, у віддалених регіонах.

Однією з основних складових успішної електронної комерції є довіра споживачів, а довіра неможлива без належного рівня безпеки. Тут на перший план виходить криптографія, яка забезпечує конфіденційність, цілісність і доступність даних, що передаються між споживачами та бізнесами. Наприклад, протоколи шифрування, такі як TLS (Transport Layer Security), захищають інформацію, яка передається між клієнтськими браузерами та серверами. Завдяки цьому споживачі можуть бути впевнені, що їхні дані, зокрема інформація про кредитні картки чи паролі, залишаються конфіденційними.

Цифрові підписи є ще одним важливим елементом криптографії в електронній комерції. Вони забезпечують цілісність транзакцій і гарантують, що дані не були змінені під час передачі. Аутентифікація, заснована на криптографічних алгоритмах, таких як RSA чи ECDSA, допомагає підтвердити особу користувачів і запобігає несанкціонованому доступу до їхніх облікових записів. Для додаткового рівня безпеки активно впроваджується двофакторна аутентифікація, яка використовує криптографічно захищені токени або одноразові паролі.

Захист платіжних даних є критично важливим аспектом електронної комерції. Більшість платіжних платформ використовують симетричне шифрування (наприклад, AES) для шифрування транзакційних даних і запобігання шахрайству. Крім того, криптографічні алгоритми допомагають відстежувати фінансові операції, запобігати їхній фальсифікації та боротися з подвійними витратами.

Криптографія забезпечує стійкість електронної комерції до сучасних кібератак, але в перспективі розвиток квантових обчислень може змінити ситуацію. Квантові комп’ютери потенційно здатні зламати сучасні алгоритми, такі як RSA чи ECDSA, що ставить під загрозу безпеку багатьох транзакцій. У зв’язку з цим важливим завданням стає перехід на пост-квантову

криптографію, яка базується на математичних задачах, стійких до квантових атак.

Таким чином, електронна комерція є невід'ємною частиною сучасного економічного ландшафту, яка відкриває нові можливості для бізнесу та споживачів. Її успішне функціонування напряму залежить від криптографії, яка забезпечує безпеку даних і захист транзакцій. З розвитком технологій важливо адаптувати криптографічні механізми до нових загроз, щоб підтримувати довіру користувачів і гарантувати стабільність електронної комерції в майбутньому.

2.1.2 Онлайн-банкінг.

Онлайн-банкінг — це система, яка дозволяє користувачам здійснювати фінансові операції через Інтернет без необхідності відвідувати фізичне відділення банку. За допомогою онлайн-банкінгу користувачі можуть керувати рахунками, переказувати кошти, оплачувати послуги, отримувати виписки, інвестувати кошти та навіть оформлювати кредити. Цей сервіс доступний через веб-сайти банків або мобільні додатки, забезпечуючи клієнтам зручність і доступність фінансових послуг у будь-який час.

Онлайн-банкінг є надзвичайно важливим у сучасному світі, оскільки він відповідає потребам цифрової економіки. Він забезпечує швидкість і ефективність фінансових операцій, зменшуючи залежність від фізичних відділень і дозволяючи користувачам отримувати доступ до своїх коштів з будь-якої точки світу. Це особливо актуально в умовах глобалізації та мобільності сучасного суспільства. Для банків онлайн-банкінг відкриває нові можливості для оптимізації роботи, зменшення витрат і залучення клієнтів.

У контексті електронної комерції онлайн-банкінг є фундаментальним елементом, оскільки забезпечує безпечні та швидкі способи оплати. Споживачі можуть миттєво переказувати кошти, оплачувати замовлення або інтегрувати свої банківські рахунки з цифровими платіжними платформами. Це сприяє розвитку електронної комерції, оскільки довіра до фінансових операцій

стимулює активність споживачів.

Криптографія відіграє ключову роль у забезпеченні безпеки онлайн-банкінгу. Протоколи шифрування, такі як TLS, захищають дані, що передаються між клієнтом і сервером, запобігаючи їх перехопленню чи модифікації. Симетричне шифрування (наприклад, AES) забезпечує збереження конфіденційної інформації, тоді як асиметричні алгоритми (RSA, ECDSA) використовуються для автентифікації та цифрових підписів. Двофакторна автентифікація та біометричні методи захисту додатково знижують ризик несанкціонованого доступу до рахунків. У перспективі розвиток квантових комп'ютерів може стати викликом для традиційних методів шифрування, тому банки вже зараз досліджують впровадження постквантових криптографічних рішень.

Таким чином, онлайн-банкінг — це не лише зручність для споживачів, але й ключовий компонент електронної комерції, де криптографія забезпечує конфіденційність, захист і довіру користувачів до цифрових фінансових операцій.

2.1.3 Комунікації (електронна пошта, обмін повідомленнями).

Сфера комунікацій, яка охоплює електронну пошту, системи обміну повідомленнями та інші цифрові способи зв'язку, є невід'ємною частиною сучасного суспільства. Ці технології забезпечують швидку та ефективну взаємодію між людьми, бізнесами та організаціями, стираючи межі між країнами та часовими зонами. Важливість цих інструментів не обмежується їхньою функціональністю; вони є основою для глобальної економіки, освіти, науки та соціальних зв'язків.

Електронна пошта — один із найстаріших і найпоширеніших способів цифрової комунікації, залишається ключовим інструментом для обміну діловою кореспонденцією, офіційними повідомленнями, маркетинговими кампаніями та особистим спілкуванням. Вона забезпечує асинхронний зв'язок, що дозволяє відправникам і отримувачам взаємодіяти у зручний для них час.

Системи обміну повідомленнями, такі як WhatsApp, Telegram, Signal або корпоративні платформи, наприклад, Microsoft Teams чи Slack, додають у комунікацію швидкість і інтерактивність, дозволяючи користувачам не лише обмінюватися текстами, але й надсилати файли, зображення, відео та організовувати відеоконференції.

У сучасному світі важливість цих інструментів зростає. Електронна пошта є обов'язковою для бізнесу, надаючи можливість зберігати офіційну документацію та організовувати робочі процеси. Системи обміну повідомленнями забезпечують гнучкість і швидкість, сприяючи кращій співпраці та координації, особливо у віддалених командах. Крім того, ці технології відкривають доступ до освітніх, медичних і соціальних послуг, що є критично важливим для громадян у віддалених регіонах або в умовах надзвичайних ситуацій, таких як пандемія.

Попри значні переваги, сфера комунікацій стикається з серйозними викликами у сфері безпеки та конфіденційності. Без належного захисту користувачі можуть стати жертвами перехоплення повідомлень, шахрайства, фішингових атак або витоку конфіденційної інформації. Тут на перший план виходить криптографія, яка є основним інструментом захисту даних у цифровій комунікації.

Для електронної пошти використання криптографії забезпечує захист не лише під час передачі, але й під час зберігання повідомлень. Протоколи, такі як TLS, шифрують дані, що передаються між клієнтом і поштовим сервером, захищаючи їх від атак типу "людина посередині". Для додаткової конфіденційності використовуються технології кінцевого шифрування, як-от PGP (Pretty Good Privacy) чи S/MIME (Secure/Multipurpose Internet Mail Extensions), які шифрують вміст листів і роблять його доступним лише для відправника та отримувача.

У системах обміну повідомленнями криптографія відіграє ще більшу роль, оскільки такі платформи часто працюють у режимі реального часу. Більшість сучасних месенджерів, наприклад Signal або WhatsApp,

використовують end-to-end encryption (E2EE), яке гарантує, що тільки учасники розмови можуть отримати доступ до її змісту. Це шифрування включає генерацію ефемерних ключів для кожної сесії, що забезпечує Perfect Forward Secrecy (PFS), завдяки чому навіть у разі компрометації одного ключа не можна дешифрувати попередні чи майбутні повідомлення.

Криптографія також використовується для автентифікації користувачів і захисту від шахрайства. Цифрові підписи дозволяють перевіряти цілісність повідомлень, а методи двофакторної автентифікації запобігають несанкціонованому доступу до облікових записів. Усе це робить комунікаційні платформи більш безпечними, що є критично важливим як для індивідуальних користувачів, так і для бізнесу.

З розвитком квантових обчислень з’являються нові виклики для захисту комунікацій, адже традиційні алгоритми, такі як RSA чи ECDSA, можуть стати вразливими до атак. Пост-квантова криптографія, яка вже розробляється, покликана забезпечити стійкість до майбутніх загроз і підтримувати високий рівень конфіденційності у сфері комунікацій.

Таким чином, сфера комунікацій є ключовою для сучасного суспільства, забезпечуючи швидкий і зручний зв’язок. Її успіх і довіра користувачів значною мірою залежать від криптографічного захисту, який гарантує конфіденційність, безпеку та цілісність даних. У світі, що постійно змінюється, роль криптографії лише зростатиме, забезпечуючи стабільність і захист цифрових комунікацій.

2.1.4 Хмарні сховища.

Хмарні сховища — це послуги, які дозволяють користувачам зберігати дані на віддалених серверах, доступних через Інтернет. Вони є однією з найбільш затребуваних технологій сучасного світу, забезпечуючи зручність, масштабованість та економічну ефективність. Завдяки хмарним сховищам користувачі можуть отримати доступ до своїх файлів з будь-якого пристрою і будь-якої точки світу, а компанії можуть зберігати та обробляти величезні

обсяги даних без необхідності в локальній інфраструктурі.

Важливість хмарних сховищ у сучасному світі зумовлена їхньою здатністю забезпечувати ефективність у роботі з даними. Для індивідуальних користувачів це можливість зберігати особисті файли, фото, відео та документи, не турбуючись про обмеження пам’яті на пристрої чи ризик втрати інформації через пошкодження обладнання. Наприклад, сервіси на зразок Google Drive, Dropbox або iCloud забезпечують резервне копіювання і синхронізацію даних, що особливо зручно для мобільних користувачів.

Для бізнесу хмарні сховища відкривають можливості для обробки великих обсягів даних, спільної роботи команд та забезпечення безперервності бізнес-процесів. Наприклад, компанії можуть використовувати хмарні платформи для спільного доступу до файлів, автоматизації резервного копіювання або швидкого масштабування, якщо обсяги даних зростають. Крім того, хмарні сервіси дозволяють значно скоротити витрати на фізичну інфраструктуру, таку як сервери та обладнання, що робить їх економічно вигідними.

Однак зі зростанням популярності хмарних сховищ зростає і значення їхньої безпеки. Оскільки дані користувачів зберігаються на віддалених серверах, важливо забезпечити їх конфіденційність, цілісність і доступність. Тут ключову роль відіграє криптографія.

Криптографія є основою безпеки хмарних сховищ. Вона захищає дані від несанкціонованого доступу, як під час їхньої передачі через Інтернет, так і під час зберігання на серверах. Протоколи шифрування, такі як TLS (Transport Layer Security), забезпечують захист даних під час передачі між клієнтським пристроєм і сервером, запобігаючи перехопленню чи модифікації інформації. Це особливо важливо для уникнення атак типу “людина посередині” (MITM).

Шифрування даних у хмарі — ще один важливий елемент захисту. Багато провайдерів використовують алгоритми симетричного шифрування, такі як AES-256, щоб гарантувати, що навіть у разі компрометації сервера зловмисники не зможуть прочитати зашифровані дані без відповідного ключа.

У деяких випадках провайдери пропонують клієнтам зберігати власні ключі шифрування, що забезпечує додатковий рівень конфіденційності, оскільки навіть сам провайдер не може отримати доступ до даних.

Криптографія також забезпечує механізми автентифікації та авторизації, які гарантують, що лише уповноважені користувачі мають доступ до даних. Наприклад, двофакторна автентифікація (2FA) додає додатковий рівень захисту, вимагаючи не лише пароль, але й одноразовий код або біометричні дані.

Одним із сучасних викликів є підготовка хмарних сховищ до квантових обчислень. Традиційні алгоритми шифрування, такі як RSA чи ECDSA, можуть бути вразливими до атак квантових комп'ютерів. Тому провайдери вже зараз досліджують і впроваджують постквантові алгоритми, такі як Kyber чи Dilithium, щоб забезпечити довгострокову стійкість до нових загроз.

Таким чином, хмарні сховища є незамінним інструментом для сучасного суспільства, забезпечуючи зручність і ефективність роботи з даними. Водночас їхній успіх та безпека напряму залежать від криптографічної складової, яка гарантує конфіденційність, цілісність і доступність даних. У майбутньому впровадження новітніх криптографічних рішень стане вирішальним фактором для підтримання довіри користувачів та адаптації до нових технологічних викликів.

2.1.5 Обмін даними в мережі.

Обмін даними в мережі є фундаментальною частиною сучасного цифрового світу, забезпечуючи передачу інформації між пристроями, системами, людьми та організаціями. Цей процес охоплює всі форми комунікації в Інтернеті, включаючи перегляд веб-сторінок, передачу файлів, потокове відео, онлайн-платежі, роботу додатків та багато іншого. Ефективний і безпечний обмін даними є основою функціонування глобальної економіки, наукових досліджень, соціальних мереж і критичних інфраструктур.

Важливість обміну даними в сучасному світі важко переоцінити. Уявіть

складність роботи сучасного суспільства без можливості миттєвої передачі інформації. Мільярди людей покладаються на ці процеси для здійснення покупок, виконання фінансових операцій, роботи чи навчання. Наприклад, системи онлайн-банкінгу, електронної комерції та комунікаційні сервіси в реальному часі базуються на безперебійному обміні даними. Бізнес використовує мережеву взаємодію для організації ланцюгів постачання, спільної роботи в командах, автоматизації процесів і аналізу великих обсягів інформації. Для наукових досліджень і розвитку технологій глобальний обмін даними дозволяє швидко обмінюватися відкриттями та співпрацювати в міжнародних масштабах.

Однак разом із масштабами і важливістю обміну даними зростають і ризики, пов’язані з безпекою та конфіденційністю. Передача даних у відкритих мережах робить їх вразливими до перехоплення, модифікації або зловмисного використання. Тут вирішальну роль відіграє криптографія, яка є основним інструментом для забезпечення безпеки та надійності обміну даними.

Криптографічна складова гарантує, що передані дані залишаються конфіденційними, цілісними та доступними лише для уповноважених сторін. Однією з основних технологій, яка забезпечує це, є TLS (Transport Layer Security). Цей протокол шифрує дані під час передачі між клієнтськими пристроями та серверами, захищаючи їх від атак типу “людина посередині” (MITM). Наприклад, під час входу на веб-сайти, що використовують HTTPS, TLS забезпечує шифрування переданих даних, таких як логіни, паролі чи номери кредитних карток.

Крім того, криптографія використовує симетричне шифрування, наприклад AES, для шифрування великих обсягів даних, і асиметричні алгоритми, такі як RSA чи ECDSA, для обміну ключами та цифрових підписів. Ці методи забезпечують безпечний обмін ключами між сторонами, навіть якщо мережа піддається моніторингу. Цифрові підписи дозволяють перевіряти, що дані не були змінені в процесі передачі, і гарантують автентичність відправника.

У сучасному світі обмін даними часто відбувається у реальному часі, наприклад, у відеоконференціях чи потоковому відео. Тут криптографія також грає ключову роль, використовуючи алгоритми з низькою затримкою для збереження швидкості передачі без шкоди для безпеки. Крім того, у випадку служб обміну файлами чи хмарних сервісів криптографія забезпечує шифрування даних не лише під час передачі, але й під час зберігання на серверах.

Зі зростанням обсягів даних і складності систем з'являються нові виклики, такі як загроза квантових обчислень. Традиційні криптографічні алгоритми, які сьогодні забезпечують захист обміну даними, можуть стати вразливими до атак квантових комп'ютерів. Тому вже зараз ведуться роботи над впровадженням пост-квантової криптографії, яка базується на задачах, стійких до квантових атак. Наприклад, алгоритми на основі решіткової криптографії, такі як Kyber, уже демонструють перспективу для заміни RSA чи ECDSA.

Таким чином, обмін даними в мережі є критично важливим для функціонування сучасного світу, забезпечуючи глобальну взаємодію, зручність і ефективність. Криптографія відіграє фундаментальну роль у захисті цих процесів, дозволяючи зберігати конфіденційність, цілісність і автентичність переданих даних. Інвестиції в сучасні криптографічні рішення, включно з постквантовими методами, є необхідністю для підтримання безпеки та довіри в цифровій епосі.

2.1.6 VPN-системи.

VPN-системи (Virtual Private Networks) — це технології, які забезпечують безпечне та конфіденційне підключення до Інтернету або приватних мереж через загальнодоступну інфраструктуру. Вони створюють зашифрований тунель між пристроєм користувача та сервером, що дозволяє захищати дані від перехоплення, приховувати реальну IP-адресу та забезпечувати доступ до ресурсів, обмежених за географічними чи

мережевими критеріями.

У сучасному світі VPN-системи набули величезної важливості. Вони дозволяють людям працювати віддалено, отримувати доступ до корпоративних мереж і захищати свої дані під час використання публічних Wi-Fi мереж, які зазвичай є вразливими до атак. З розвитком концепції віддаленої роботи VPN стали невід'ємним інструментом для бізнесу, забезпечуючи працівникам захищений доступ до внутрішніх ресурсів компанії, таких як бази даних, файлові системи та сервіси. VPN також використовуються для обходу цензури в країнах із обмеженим доступом до Інтернету, забезпечуючи свободу слова та можливість доступу до інформації.

Однак важливість VPN виходить за рамки приватності та доступу. Вони також є ключовим інструментом для захисту конфіденційності користувачів в умовах постійно зростаючих загроз у кіберпросторі, таких як перехоплення даних, спостереження або цілеспрямовані атаки. Саме тут на передній план виходить криптографія, яка є основою безпеки VPN-систем.

Криптографічна складова VPN забезпечує шифрування даних, автентифікацію користувачів і захист від атак. Основним механізмом є створення захищеного тунелю між клієнтом і сервером VPN, що шифрує весь трафік, який проходить через нього. Протоколи, такі як OpenVPN, IKEv2/IPSec, WireGuard та L2TP/IPSec, забезпечують надійне шифрування переданих даних і захищають їх від перехоплення. Для шифрування зазвичай використовуються алгоритми, такі як AES-256, які забезпечують високий рівень захисту навіть від сучасних атак.

Криптографія також грає вирішальну роль у процесі автентифікації. Коли клієнт підключається до VPN-сервера, використовуються асиметричні алгоритми, наприклад, RSA чи ECDSA, для обміну ключами, які забезпечують безпечне встановлення з'єднання. Ці ключі використовуються для шифрування трафіку та захисту від атак типу "людина посередині" (MITM). Додатково цифрові сертифікати та цифрові підписи гарантують, що користувач підключається до легітимного сервера, а не до підробленого.

Для багатьох користувачів VPN важливою функцією є приховування їхньої реальної IP-адреси. Це дозволяє зберігати анонімність в Інтернеті та уникати стеження з боку провайдерів чи державних органів. Тут криптографія забезпечує не лише захист даних, але й гарантує, що маршрутизація трафіку відбувається конфіденційно та без ризику його відстеження.

Зі зростанням потужності квантових обчислень з'являється загроза для традиційних криптографічних алгоритмів, які використовуються у VPN. Наприклад, RSA чи ECDSA можуть бути вразливими до алгоритму Шора, реалізованого на квантовому комп'ютері. Для захисту від таких загроз вже розробляються постквантові криптографічні алгоритми, такі як Kyber чи Dilithium, які можуть бути інтегровані в протоколи VPN для забезпечення їхньої довгострокової стійкості.

Таким чином, VPN-системи є критично важливими для забезпечення конфіденційності, безпеки та доступу до інформації у сучасному світі. Їхній успіх напряму залежить від криптографічної складової, яка забезпечує захист даних, цілісність з'єднання та автентифікацію користувачів. У майбутньому інтеграція постквантової криптографії стане ключовим фактором для адаптації VPN до нових викликів, забезпечуючи захист навіть у світі, де квантові обчислення стануть реальністю.

2.1.7 Блокчейн.

Блокчейн — це децентралізована технологія зберігання та обміну даними, яка функціонує як цифровий реєстр транзакцій, організованих у вигляді послідовно з'єднаних блоків. Ця технологія стала основою для криптовалют, таких як Bitcoin і Ethereum, але її застосування виходить далеко за межі фінансової сфери. Блокчейн характеризується прозорістю, незмінністю даних і децентралізацією, що робить його унікальним інструментом для забезпечення довіри та безпеки в різних галузях.

Важливість блокчейну у сучасному світі постійно зростає. Він

трансформує фінансову систему, усуваючи необхідність у посередниках, таких як банки чи платіжні системи, і забезпечуючи пряму взаємодію між учасниками. Це відкриває доступ до фінансових послуг для мільйонів людей, які раніше не мали до них доступу. Крім того, блокчейн сприяє автоматизації бізнес-процесів через смарт-контракти — програми, що автоматично виконують угоди відповідно до заданих умов. У логістиці він забезпечує відстеження походження товарів і прозорість постачання. У сфері охорони здоров'я блокчейн використовується для захисту даних пацієнтів і управління електронними медичними записами.

Однією з головних причин успіху блокчейну є його здатність забезпечувати довіру між учасниками системи без необхідності у центральному органі. Це досягається за допомогою криптографії, яка є фундаментальною складовою блокчейн-технології. Криптографія гарантує безпеку, цілісність і автентичність даних, що зберігаються у блокчейні.

По-перше, криптографія використовується для створення цифрових підписів, які забезпечують автентифікацію транзакцій. У більшості блокчейнів, таких як Bitcoin, для цього використовується алгоритм ECDSA (Elliptic Curve Digital Signature Algorithm). Коли користувач ініціює транзакцію, його приватний ключ використовується для створення підпису, який може бути перевірений іншими учасниками мережі за допомогою відповідного публічного ключа. Це гарантує, що транзакція була ініційована уповноваженим користувачем і не була змінена після створення.

По-друге, для забезпечення цілісності даних використовується хешування. Кожен блок у ланцюгу містить хеш попереднього блоку, що створює криптографічний зв'язок між блоками. Якщо дані в одному з блоків змінюються, це автоматично змінює хеш наступних блоків, роблячи будь-яку спробу модифікації очевидною для всіх учасників. Алгоритми хешування, такі як SHA-256, забезпечують надійний захист від підробки даних.

По-третє, криптографія використовується для консенсусу — механізму, який дозволяє учасникам мережі дійти згоди щодо валідності транзакцій.

Наприклад, у Bitcoin використовується Proof of Work, де обчислювальні потужності учасників мережі використовуються для розв’язання криптографічних задач. Інші блокчейни, такі як Ethereum 2.0, переходять до Proof of Stake, який зменшує енергоспоживання та залежність від обчислювальних ресурсів.

Блокчейн також стикається з викликами, пов’язаними з майбутнім розвитком квантових комп’ютерів. Алгоритми, такі як ECDSA та RSA, які широко використовуються для цифрових підписів і шифрування, можуть бути вразливими до квантових атак. Це стимулює дослідження пост-квантової криптографії, яка здатна забезпечити стійкість до таких загроз. Наприклад, алгоритми на основі решіток або хеш-функцій, такі як SPHINCS+ чи Dilithium, розглядаються як можливі замінники класичних методів.

Таким чином, блокчейн є важливою технологією, яка змінює спосіб організації та ведення бізнесу, забезпечуючи прозорість, безпеку та автоматизацію процесів. Криптографія є серцем блокчейн-системи, забезпечуючи її ключові характеристики: децентралізацію, незмінність і довіру. У майбутньому впровадження пост-квантових алгоритмів стане важливим етапом для захисту блокчейну від нових загроз, зберігаючи його позиції як одного з найнадійніших інструментів у цифрову епоху.

2.2 Дослідження та аналіз сучасних криптографічних методів.

2.2.1 Симетричні методи.

Симетричні методи криптографії — це способи шифрування, у яких один і той самий ключ використовується для шифрування та дешифрування даних. Цей підхід є одним із найстаріших і найшвидших методів забезпечення конфіденційності інформації, і він досі широко застосовується завдяки своїй ефективності.

Основна ідея симетричної криптографії полягає в тому, що відправник і отримувач повинні заздалегідь обмінятися спільним секретним ключем, який потім використовується для обробки даних. Алгоритми симетричного

шифрування забезпечують високу швидкість роботи й мале навантаження на обчислювальні ресурси.

Однак головною вразливістю є необхідність безпечного обміну ключами між сторонами, що може бути проблематичним у великих або незахищених мережах. Для вирішення цієї проблеми симетричні методи часто поєднуються з асиметричною криптографією, яка використовується для безпечної передачі ключів.

2.2.1.1 Важливість та ефективність симетричних алгоритмів шифрування.

Виконуючи операції шифрування та дешифрування за допомогою одного і того ж ключа, симетричні методи працюють швидше, ніж асиметричні алгоритми, що використовують пару ключів. Ця ефективність особливо важлива в системах, де обробляються великі обсяги даних у режимі реального часу, таких як відеостріми, захищені з'єднання в Інтернеті чи системи зберігання даних.

Широке розповсюдження симетричних алгоритмів пояснюється їхньою універсальністю. Вони застосовуються в багатьох галузях: від фінансових транзакцій і VPN-з'єднань до захисту конфіденційної інформації в корпоративних мережах та мобільних додатках. Наприклад, вони використовуються для захисту комунікацій між пристроями, забезпечення безпеки збережених даних і навіть для захисту інформації, переданої через супутники чи IoT-пристрої. Завдяки простоті впровадження й мінімальним вимогам до ресурсів, симетричні методи залишаються ключовими в системах, де швидкість і економія ресурсів є пріоритетними.

Ефективність симетричних алгоритмів базується на їхньому математичному фундаменті, що дозволяє створювати стійке до атак шифрування. Однак важливо відзначити, що їхня безпека залежить від конфіденційності ключа. Якщо ключ потрапляє до зловмисників, вони можуть отримати доступ до всієї зашифрованої інформації. Це є головною вразливістю симетричних алгоритмів. Для подолання цієї проблеми симетричні методи

часто використовуються у поєднанні з асиметричними, які забезпечують безпечну передачу ключів.

Розвиток технологій зберігає за симетричними алгоритмами їхню актуальність. У системах із високими вимогами до продуктивності та безпеки вони залишаються основним інструментом, здатним забезпечити баланс між швидкістю роботи та криптографічною стійкістю. Завдяки адаптації до нових вимог, таких як інтеграція з постквантовими методами, симетричні алгоритми продовжують залишатися важливим компонентом глобальної системи кібербезпеки.

2.2.1.2 Математичне обґрунтування.

Симетричні методи криптографії можна пояснити математично простим прикладом. Уявімо, що дані, які потрібно захистити - це ваше повідомлення (назвемо його M), а ключ — це секретний код, який допомагає шифрувати і дешифрувати це повідомлення (назвемо його K).

Шифрування.

Симетричний алгоритм бере ваше повідомлення M і ключ K, а потім застосовує математичну функцію E (наприклад, перестановки, підстановки чи операції XOR), щоб створити зашифроване повідомлення (шифротекст C).

$$C = E\ (M, K) \tag{1}$$

де E - це алгоритм шифрування.

Як приклад, функція може взяти кожну літеру в повідомленні й змістити її в алфавіті на значення ключа K. Наприклад, якщо K = 3 , літера “A” стає “D”, “B” стає “E”, і так далі.

Дешифрування.

Для отримання початкового повідомлення з шифротексту, інша сторона

використовує той самий ключ K і обернену функцію D.

$$M = D\ (C, K) \tag{2}$$

Тобто, якщо E - це зміщення на K, то D буде зміщення назад на K.

2.2.1.3 Сценарії використання симетричних алгоритмів.

Однією з ключових сфер застосування симетричних алгоритмів є фінансові системи. Банківські транзакції, системи обробки платежів, онлайн-банкінг та мобільні платіжні додатки активно використовують ці методи для захисту конфіденційної інформації, такої як номери банківських карт, паролі чи деталі рахунків. Наприклад, під час здійснення транзакцій через POS-термінали дані шифруються симетричними методами для запобігання їх перехопленню зловмисниками. Швидкість симетричних алгоритмів дозволяє обробляти мільйони транзакцій на секунду, забезпечуючи безперебійність роботи фінансових систем.

У сфері мережевих з'єднань симетричні алгоритми відіграють центральну роль у захисті переданих даних. Вони використовуються в VPN-з'єднаннях для створення захищеного тунелю між клієнтом і сервером. Завдяки цьому користувачі можуть безпечно передавати дані через загальнодоступні мережі, наприклад, у громадських Wi-Fi. Шифрування трафіку за допомогою симетричних алгоритмів захищає дані від перехоплення й модифікації під час їхньої передачі.

Симетричні алгоритми також є основою безпеки у протоколах HTTPS, які використовуються для захищеного доступу до веб-сайтів. Після встановлення сесійного ключа за допомогою асиметричного шифрування, саме симетричні алгоритми використовуються для швидкого шифрування даних, що передаються між клієнтом і сервером. Це забезпечує захист паролів, платіжних даних та іншої конфіденційної інформації під час онлайн-

комунікацій.

Ще однією важливою сферою використання є зберігання даних. Симетричні алгоритми застосовуються для шифрування файлів і баз даних у корпоративних середовищах, захищаючи їх навіть у разі фізичного доступу до носіїв. У хмарних сховищах вони забезпечують конфіденційність даних під час зберігання, а також захист від несанкціонованого доступу.

У сфері мобільних додатків і пристроїв Інтернету речей (IoT) симетричні алгоритми є критично важливими завдяки їхній енергоефективності та низьким вимогам до обчислювальних ресурсів. Наприклад, у месенджерах симетричне шифрування забезпечує конфіденційність повідомлень у реальному часі, а у смарт-пристроях — захист даних і команд між пристроями.

Симетричні алгоритми також застосовуються в стримінгових сервісах, де вони забезпечують шифрування відео- чи аудіо-контенту, захищаючи його від піратства або несанкціонованого доступу. Крім того, у системах контролю доступу вони шифрують дані, які використовуються для ідентифікації користувачів.

2.2.1.4 Сучасні алгоритми симетричного шифрування.

2.2.1.4.1 AES

AES є найпоширенішим і найбільш надійним алгоритмом симетричного шифрування на сьогодні. Він був затверджений як стандарт шифрування урядом США в 2001 році і замінив застарілий DES. AES працює з блоками розміром 128 біт і підтримує ключі довжиною 128, 192 або 256 біт.

Принцип роботи.

Принцип роботи AES базується на підстановочно-перестановочній мережі (SPN), яка забезпечує стійкість до атак завдяки комбінації нелінійних підстановок, лінійних перестановок та операцій XOR.

На початковому етапі дані, які потрібно зашифрувати, розбиваються на блоки по 128 біт. Якщо обсяг даних не кратний розміру блоку, застосовуються

методи доповнення, щоб заповнити відсутні біти. Для шифрування використовується ключ, який проходить через процес розкладу, щоб створити набір раундових ключів. Кількість раундів залежить від довжини ключа: для 128-бітного ключа проводиться 10 раундів, для 192-бітного — 12, а для 256-бітного — 14.

Шифрування починається з початкової операції XOR між блоком даних і першим раундовим ключем. Потім дані проходять через основний цикл, який включає кілька послідовних кроків: SubBytes, ShiftRows, MixColumns і AddRoundKey. На етапі SubBytes кожен байт у блоці замінюється на новий за допомогою підстановочної таблиці (S-Box), яка забезпечує нелінійність шифрування. Після цього виконується ShiftRows — циклічна перестановка рядків блоку, яка порушує зв’язок між сусідніми байтами. На етапі MixColumns кожен стовпець блоку проходить через лінійне перетворення з використанням множення в полях Галуа, що забезпечує поширення впливу змін. Завершується кожен раунд операцією AddRoundKey, де до блоку даних додається поточний раундовий ключ за допомогою операції XOR.

Останній раунд шифрування відрізняється від попередніх, оскільки в ньому пропускається етап MixColumns. Це гарантує, що структура даних не ускладнюється зайвими операціями, але при цьому зберігається високий рівень безпеки. Після завершення всіх раундів формується зашифрований блок, який потім об’єднується з іншими блоками для отримання фінального шифротексту.

Процес дешифрування є зворотним до шифрування. Він використовує ті самі раундові ключі в зворотному порядку та виконує обернені операції для кожного етапу: InvSubBytes, InvShiftRows, InvMixColumns і AddRoundKey. Це дозволяє відновити початкові дані без втрати інформації, якщо ключ залишився незмінним.

Сфери застосування.

У мережевих з’єднаннях, таких як VPN і HTTPS, він забезпечує

конфіденційність і захист переданих даних. Хмарні сервіси використовують AES для шифрування файлів, резервних копій та баз даних. Мобільні додатки й пристрої IoT покладаються на AES через його ефективність на обмежених ресурсах. У фінансових системах AES захищає транзакції та платіжні дані, а у військових і урядових мережах — конфіденційну інформацію. Він є стандартом для безпеки даних у сучасному цифровому світі.

Переваги:

а) Висока швидкість і продуктивність.

AES оптимізований для швидкої роботи як у програмних, так і в апаратних реалізаціях. Завдяки цьому він працює швидше, ніж алгоритми, такі як Twofish або Camellia, особливо коли використовується апаратне прискорення, наприклад, через AES-NI (інструкції Intel).

б) Широка стандартизація та підтримка.

AES є затвердженим стандартом шифрування, прийнятим Національним інститутом стандартів і технологій США (NIST). Завдяки цьому він підтримується майже в усіх сучасних протоколах, таких як TLS, IPSec і SSH, а також у хмарних і мобільних платформах.

в) Безпека на високому рівні.

AES використовує підстановочно-перестановочну мережу, яка забезпечує стійкість до основних типів атак, включаючи криптоаналітичні (диференційний та лінійний криптоаналіз). Для AES-256 на сьогодні не існує ефективних атак, які б ставили під загрозу його безпеку.

г) Гнучкість у виборі ключів.

AES підтримує кілька розмірів ключів (128, 192, 256 біт), що дозволяє налаштовувати рівень безпеки залежно від потреб. Наприклад, AES-128 забезпечує баланс між швидкістю та безпекою, тоді як AES-256 надає максимальний захист.

ґ) Відносно невеликий розмір ключів.

Порівняно з деякими іншими алгоритмами, AES забезпечує високу безпеку без надмірного збільшення розміру ключів. Це особливо важливо для обмежених систем.

Недоліки:

а) Висока залежність від апаратного прискорення.

Без апаратної підтримки AES може працювати повільніше, ніж алгоритми, такі як ChaCha20, які оптимізовані для програмних реалізацій. Це особливо актуально для мобільних пристроїв і серверів із обмеженими ресурсами.

б) Складність реалізації.

Через складну структуру раундів (SubBytes, ShiftRows, MixColumns) реалізація AES вимагає ретельного налаштування, щоб уникнути вразливостей, таких як побічні атаки (side-channel attacks), що експлуатують час виконання або споживання енергії.

в) Менша ефективність для потокового шифрування.

Як блочний алгоритм, AES менш ефективний для потокового шифрування, ніж ChaCha20 або RC4. Для потокових даних його зазвичай використовують у режимах роботи, наприклад, CTR, але це збільшує складність реалізації.

г) Ресурсоємність на пристроях із обмеженими можливостями.

На пристроях з обмеженою обчислювальною потужністю AES може поступатися ChaCha20, який забезпечує аналогічний рівень безпеки при меншому навантаженні на процесор.

ґ) Складність у інтеграції в пост-квантові системи.

Хоча AES вважається стійким до квантових атак при використанні

великих ключів (AES-256), для нього важче інтегрувати сучасні механізми, що адаптують його до повністю пост-квантового середовища, порівняно з новими алгоритмами.

2.2.1.4.2 ChaCha20

ChaCha20 - це сучасний потоковий алгоритм симетричного шифрування, який забезпечує високу швидкість і безпеку. Він використовує псевдовипадковий потік даних, який XOR-иться з відкритим текстом для створення шифротексту. Основою алгоритму є послідовність простих операцій: додавання, XOR і циклічні зрушення, що робить його стійким до атак і ефективним навіть на пристроях із обмеженими ресурсами. ChaCha20 не потребує апаратного прискорення, на відміну від AES, і забезпечує стабільну продуктивність у програмному забезпеченні.

Принцип роботи.

Основою алгоритму є функція ChaCha, яка працює з 512-бітним станом, поділеним на 16 слів по 32 біти. Вхідні параметри складаються з ключа (256 біт), унікального номера блоку, 96-бітного одноразового значення (nonce) і постійних слів.

ChaCha20 виконує 20 раундів обчислень, у яких використовуються прості операції: додавання за модулем 2^{32}, побітова операція XOR і циклічні зрушення. Ці раунди включають серію перетворень, що змішують значення в масиві, створюючи високоентропійний вихід. Кожен блок генерує 64 байти ключового потоку, які XOR-яться з відповідною частиною відкритого тексту. Завдяки цьому ChaCha20 забезпечує високу швидкість шифрування та стійкість до атак, таких як диференціальний криптоаналіз.

На відміну від AES, який є блочним алгоритмом і працює з фіксованими блоками 128 біт із підстановочно-перестановочними мережами, ChaCha20 є потоковим і генерує псевдовипадковий потік даних. ChaCha20 використовує прості арифметичні операції, що робить його швидшим і менш залежним від апаратного прискорення, на відміну від AES, який оптимізований для роботи

на апаратному рівні, особливо за допомогою AES-NI.

Сфери застосування.

ChaCha20 активно використовується в різних сферах завдяки своїй високій швидкості, стійкості до атак і ефективності на пристроях із обмеженими ресурсами. У протоколі TLS 1.3 ChaCha20 застосовується для шифрування даних у мережевих з'єднаннях, забезпечуючи безпеку передачі інформації, особливо на мобільних пристроях. Він також широко використовується у VPN-з'єднаннях, де потрібна швидка й надійна обробка трафіку без високого навантаження на процесор. У мобільних додатках, таких як месенджери чи фінансові сервіси, ChaCha20 є оптимальним вибором завдяки своїй стабільній роботі навіть без апаратного прискорення.

Переваги:

а) Швидкість.

ChaCha20 забезпечує високу швидкість шифрування навіть на пристроях без апаратного прискорення. Це досягається завдяки використанню простих операцій, таких як додавання, XOR і циклічні зрушення, які ефективно виконуються на будь-якому процесорі. На програмному рівні ChaCha20 часто перевершує AES, особливо на мобільних пристроях.

б) Безпека.

ChaCha20 має високу стійкість до криптоаналітичних атак, включно з диференціальним і лінійним криптоаналізом. Його архітектура унеможливлює поширені атаки, які характерні для RC4, і робить його надійним вибором навіть для критичних систем безпеки.

в) Ефективність.

Завдяки простоті своїх операцій ChaCha20 споживає менше ресурсів на пристроях із низькою обчислювальною потужністю, таких як IoT-девайси та мобільні телефони. Це робить його ідеальним для використання в сценаріях із

обмеженими ресурсами.

г) Простота реалізації.

Алгоритм має просту структуру і не потребує складних таблиць чи підстановок (на відміну від AES із його S-Box). Це знижує ймовірність помилок під час реалізації та робить алгоритм зручним для впровадження в різних програмних середовищах.

Недоліки:

а) Підтримка.

ChaCha20 не має такої широкої підтримки у старих системах і протоколах, як AES, який став стандартом для багатьох мережевих протоколів (наприклад, TLS). Це може обмежувати його застосування у вже існуючих інфраструктурах.

б) Енергоємність.

Хоча ChaCha20 ефективний у програмних реалізаціях, він не може використовувати апаратне прискорення, таке як AES-NI. У результаті AES, що працює на сучасному апаратному обладнанні, може споживати менше енергії в довготривалих сесіях.

в) Розмір ключового потоку.

Як потоковий алгоритм, ChaCha20 залежить від синхронізації між відправником і отримувачем. Якщо синхронізація порушується, навіть невелика помилка може призвести до втрати всієї сесії або до того, що дані стануть нерозшифровуваними.

2.2.1.4.3 Camellia

Camellia — це сучасний блочний симетричний алгоритм шифрування, розроблений у Японії, який забезпечує високий рівень безпеки, еквівалентний AES. Він працює з блоками даних розміром 128 біт і підтримує ключі

довжиною 128, 192 і 256 біт. Camellia використовує структуру Feistel із 18 або 24 раундами (залежно від довжини ключа) і застосовує нелінійні перетворення, підстановки, перестановки та XOR. Алгоритм оптимізований для програмних і апаратних реалізацій, демонструючи високу швидкість і енергоефективність.

Принцип роботи.

Алгоритм складається з кількох основних операцій, які чергуються між раундами: нелінійні перетворення (S-Box), перестановки (P-Box) і побітові операції XOR. Дані поділяються на дві частини: ліва обробляється, права залишається незмінною, після чого сторони міняються місцями. Завдяки такій структурі Camellia забезпечує ефективне поширення змін ключа та даних через кілька раундів, що створює високу стійкість до атак, таких як диференціальний і лінійний криптоаналіз. Алгоритм також включає унікальні перетворення FL і FL^{-1}, які підвищують рівень безпеки.

Camellia найбільш схожий на AES, оскільки обидва алгоритми працюють із блоками даних однакового розміру і підтримують ключі до 256 біт. Проте, на відміну від підстановочно-перестановочної структури AES, Camellia базується на Feistel-мережі, що спрощує її зворотність і робить реалізацію дешифрування легшою. Camellia також включає унікальні елементи, такі як FL-операції, які відсутні в AES, що надає алгоритму власні характеристики захисту.

Сфери застосування.

Camellia використовується в багатьох сферах, де необхідна висока безпека, ефективність і відповідність міжнародним стандартам. У корпоративних системах він забезпечує шифрування конфіденційних даних, таких як фінансова інформація, документи та бази даних, гарантуючи їх захист навіть у разі фізичного доступу до серверів. Урядові організації та військові структури використовують Camellia для захисту секретної інформації завдяки його схваленню міжнародними стандартами безпеки, такими як ISO/IEC.

Camellia також знаходить застосування в фінансових системах для

шифрування транзакцій і захисту платіжних даних. У мережевих протоколах, таких як TLS/SSL та IPsec, алгоритм забезпечує безпеку трафіку, що передається через Інтернет. Завдяки своїй універсальності та швидкості, Camellia використовується в захисті мобільних додатків, де важлива оптимізація продуктивності на обмежених ресурсах. Хмарні сервіси та системи резервного копіювання застосовують Camellia для збереження даних у зашифрованому вигляді, знижуючи ризики несанкціонованого доступу.

Переваги:

а) Висока безпека.

Camellia має стійкість до сучасних криптоаналітичних атак, таких як диференціальний, лінійний криптоаналіз і атаки з використанням розкладу ключів. Завдяки використанню складних раундових операцій, включно з FL- та FL^{-1}-перетвореннями, алгоритм забезпечує надійний захист даних навіть у тривалих сесіях.

б) Гнучкість.

Camellia підтримує кілька довжин ключів: 128, 192 і 256 біт, що дозволяє налаштувати рівень безпеки відповідно до потреб конкретного застосування. Це робить його універсальним для різних сценаріїв, включно з високобезпечними та ресурсозалежними середовищами.

в) Продуктивність.

Алгоритм ефективний як у програмних реалізаціях, так і на апаратному рівні. Завдяки своїй Feistel-структурі він забезпечує оптимальну продуктивність для шифрування та дешифрування, що робить його придатним для використання на пристроях із обмеженими ресурсами.

г) Сертифікація.

Camellia відповідає міжнародним стандартам безпеки, включно з ISO/IEC, що забезпечує його надійність і прийняття в урядових, військових і

корпоративних системах. Така сертифікація свідчить про його ефективність та придатність для захисту конфіденційних даних.

Недоліки:

а) Непопулярність.

Camellia не має такої широкої підтримки, як AES, у популярних протоколах і системах. Наприклад, не всі програмні бібліотеки та платформи реалізують його, що може ускладнити інтеграцію в уже існуючі системи.

б) Складність реалізації.

Використання унікальних елементів, таких як FL-операції, вимагає ретельної реалізації, оскільки навіть незначні помилки можуть створити вразливості. Це може ускладнити впровадження алгоритму порівняно з іншими, простішими у реалізації алгоритмами.

в) Енергоємність.

Camellia споживає більше обчислювальних ресурсів порівняно з легкими алгоритмами, такими як ChaCha20. У високонавантажених середовищах або на пристроях із дуже обмеженими ресурсами це може бути проблемою, якщо енергоспоживання є критичним фактором.

2.2.1.4.4 Twofish

Twofish — це блочний симетричний алгоритм шифрування, розроблений як фіналіст конкурсу AES. Він працює з блоками розміром 128 біт і підтримує ключі довжиною до 256 біт. Основою Twofish є модифікована структура Файстеля з використанням динамічних S-Box, що генеруються на основі ключа. Це забезпечує високу гнучкість і стійкість до атак. Алгоритм використовує прості арифметичні операції, такі як XOR і підстановки, що забезпечує ефективність у програмних і апаратних реалізаціях.

Принцип роботи.

Алгоритм починається з розкладу ключа, який генерує кілька підключів для кожного раунду, а також ключі для S-Box. Це робить Twofish надзвичайно динамічним і стійким до атак, що базуються на аналізі фіксованих структур.

Блоки даних поділяються на дві частини, які проходять через 16 раундів обробки. Кожен раунд включає нелінійні перетворення (S-Box), перестановки та побітові операції XOR. Унікальною особливістю Twofish є використання динамічних S-Box, які генеруються на основі ключа, що ускладнює аналіз структури алгоритму. Додатково застосовуються матричні множення в модульному полі, що підсилює поширення змін у даних. Після всіх раундів виконується об’єднання двох частин блоку для отримання шифротексту. Завдяки своїй гнучкій конструкції Twofish підходить для апаратних і програмних реалізацій.

Twofish є наступником Blowfish, і обидва алгоритми базуються на структурі Файстеля. Однак, на відміну від Blowfish із фіксованими S-Box, Twofish використовує динамічні S-Box, що генеруються з ключа. Це робить Twofish безпечнішим і придатнішим для використання з довгими ключами. Twofish також є ефективнішим у сучасних програмних реалізаціях.

Сфери застосування.

Twofish застосовується в різних сферах, де потрібне ефективне та надійне шифрування. У системах зберігання даних алгоритм використовується для захисту файлів, баз даних і резервних копій, забезпечуючи конфіденційність навіть у разі втрати доступу до фізичних носіїв. Він також популярний у програмному забезпеченні з відкритим кодом, зокрема для шифрування файлів і дисків, таких як програми TrueCrypt і VeraCrypt.

У корпоративних середовищах Twofish використовується для захисту конфіденційних даних і внутрішньої комунікації завдяки високій безпеці та гнучкості. У мобільних додатках і IoT-пристроях алгоритм застосовується через свою ефективність у програмних реалізаціях, що важливо для пристроїв із обмеженими ресурсами. Twofish також підходить для фінансових систем, де

потрібен захист транзакцій та даних клієнтів. Його безпека та гнучкість роблять його придатним для шифрування в умовах із підвищеними вимогами до захисту.

Переваги:

а) Висока безпека.

Завдяки динамічним S-Box, які генеруються на основі ключа, і складному розкладу ключів, Twofish має високу стійкість до атак, таких як диференціальний та лінійний криптоаналіз. Крім того, алгоритм використовує інші елементи, такі як перестановки та матричні множення, що підвищують його криптографічну стійкість.

б) Гнучкість.

Twofish підтримує ключі довжиною до 256 біт, що дозволяє налаштовувати рівень безпеки залежно від потреб. Для додатків із меншими вимогами можна використовувати коротші ключі, зберігаючи при цьому швидкість роботи.

в) Ефективність.

Алгоритм оптимізований для роботи в різних середовищах, як програмних, так і апаратних. Він демонструє гарну продуктивність навіть на пристроях із обмеженими ресурсами, таких як IoT-пристрої чи мобільні телефони. Його гнучка конструкція дозволяє інтегрувати Twofish у широкий спектр платформ.

г) Відкритий код.

Twofish є алгоритмом із відкритим вихідним кодом, що забезпечує його широке використання у програмному забезпеченні, такому як VeraCrypt. Це також сприяє його доступності для аудиту безпеки.

Недоліки:

а) Непопулярність.

Twofish не став таким популярним, як AES, оскільки AES було обрано як офіційний стандарт шифрування (NIST). Через це багато бібліотек, протоколів і пристроїв не підтримують Twofish, що обмежує його застосування в індустрії.

б) Складність реалізації.

Реалізація динамічних S-Box і матричних операцій вимагає значної уваги до деталей. Неправильна реалізація може створити вразливості, що робить його менш привабливим для недосвідчених розробників.

в) Продуктивність.

У порівнянні з AES, Twofish може бути трохи повільнішим, особливо в апаратних реалізаціях, де AES часто має перевагу завдяки апаратному прискоренню (AES-NI). У середовищах із великим навантаженням це може стати критичним фактором.

2.2.2 Асиметричні методи.

Асиметричні методи криптографії базуються на використанні пари ключів: відкритого (публічного) і закритого (приватного). Відкритий ключ використовується для шифрування даних або перевірки цифрових підписів, тоді як приватний ключ — для дешифрування або створення підписів. Головна особливість цих методів полягає в тому, що знання відкритого ключа не дозволяє отримати приватний, що забезпечує високий рівень безпеки.

Асиметричні алгоритми, такі як RSA, ECDSA і Diffie-Hellman, широко використовуються для захисту конфіденційної інформації в мережевих з’єднаннях (TLS), обміну ключами та цифрових підписів. Наприклад, при встановленні захищеного з’єднання відкритий ключ сервера передається клієнту, щоб клієнт міг шифрувати дані для надсилання.

Хоча асиметричні методи забезпечують високий рівень захисту, вони є обчислювально більш затратними, ніж симетричні, що обмежує їх

використання для великих обсягів даних. Зазвичай асиметричні алгоритми застосовуються для початкового обміну ключами, після чого для шифрування основного трафіку використовується симетричний метод.

2.2.2.1 Важливість та ефективність асиметричних алгоритмів шифрування.

Асиметричні алгоритми криптографії є однією з найважливіших складових сучасної кібербезпеки. Їхня унікальність полягає у використанні пари ключів: відкритого, який може бути загальнодоступним, і приватного, який зберігається у таємниці. Це дозволяє забезпечувати безпеку даних навіть у відкритих мережах, таких як Інтернет, де інформація може бути доступна для перехоплення.

Асиметричні алгоритми мають критичне значення для багатьох аспектів цифрового світу. Вони забезпечують захищений обмін даними, автентифікацію користувачів і підписання документів. Їхня основна роль — гарантування конфіденційності передачі інформації, цілісності даних і перевірки автентичності учасників комунікації. Завдяки цьому асиметрична криптографія є основою для роботи сучасних протоколів, таких як TLS, які використовуються для захищених веб-з’єднань, і PKI (інфраструктура відкритих ключів), яка підтримує цифрові сертифікати.

Розповсюдження асиметричних алгоритмів охоплює практично всі галузі, де потрібна безпека даних. Від фінансових систем і електронної комерції до хмарних обчислень і мобільних додатків — їх використовують скрізь, де необхідний захист конфіденційної інформації. Асиметричні методи особливо важливі в системах, які потребують безпечного обміну ключами для шифрування трафіку, наприклад, у VPN, або для створення цифрових підписів, які гарантують достовірність транзакцій у блокчейн-технологіях.

Ефективність асиметричних алгоритмів залежить від їхньої здатності забезпечувати високий рівень безпеки, навіть коли відкритий ключ знаходиться у вільному доступі. Їхні математичні основи, такі як факторизація великих чисел або обчислення дискретного логарифма, забезпечують

складність зламу приватного ключа, що робить ці алгоритми надзвичайно надійними. Однак вони вимагають значних обчислювальних ресурсів, що є їхнім головним недоліком у порівнянні з симетричними алгоритмами.

Незважаючи на це, асиметричні методи залишаються ефективними завдяки їхній ролі у встановленні сесійних ключів для симетричних алгоритмів. Після встановлення безпечного з'єднання за допомогою асиметричної криптографії для основного шифрування даних використовується швидший симетричний алгоритм. Така комбінація забезпечує баланс між безпекою та продуктивністю.

У майбутньому актуальність асиметричних алгоритмів залишиться високою, хоча розвиток квантових обчислень ставить нові виклики перед їхньою стійкістю. Це стимулює дослідження в галузі постквантової криптографії, яка буде здатна забезпечити аналогічний рівень безпеки в нових умовах. Таким чином, асиметрична криптографія є незамінною технологією для сучасної та майбутньої цифрової інфраструктури.

2.2.2.2 Математичне обґрунтування.

Асиметричні методи криптографії базуються на математичних задачах, які легко виконати в одному напрямку, але майже неможливо повернути назад без спеціальної інформації (наприклад, закритого ключа).

Генерація ключів.

Для реалізації асиметричного алгоритму шифрування потрібно згенерувати 2 ключі, на відміну від симетричних алгоритмів:

- приватний ключ, що тримається в секреті та не передається через незахищені канали;
- публічний ключ, який повинен бути переданий.

Ці ключі пов'язані через математичну функцію, наприклад, множення великих чисел або обчислення на основі еліптичних кривих.

Шифрування.

Коли хтось хоче відправити вам повідомлення, він бере ваш публічний ключ public_key і шифрує своє повідомлення M за допомогою цього ключа через математичну операцію.

$$C = E\ (M, public\_key) \tag{3}$$

де E - це алгоритм шифрування;

C - шифротекст.

Важливо, що без приватного ключа дешифрувати C неможливо.

Дешифрування.

Тільки ви, маючи свій приватний ключ private_key, можете розшифрувати шифротекст.

$$M = D\ (C, private\_key) \tag{4}$$

де D - це алгоритм дешифрування.

Цифровий підпис.

У випадку цифрових підписів усе працює навпаки: ви створюєте підпис за допомогою приватного ключа.

$$S = Sign\ (M, private\_key) \tag{5}$$

де Sign - алгоритм підпису.

Отримувач перевіряє цей підпис, використовуючи ваш публічний ключ.

$$R = Verify\ (S, M, public\_key) \tag{6}$$

де Verify – алгоритм верифікації пидпису;

R – результат верифікації (true, або false).

2.2.2.3 Сценарії використання асиметричних алгоритмів.

Асиметричні алгоритми криптографії мають широкий спектр застосувань у сучасному світі завдяки своїй здатності забезпечувати захист даних, автентифікацію, конфіденційність і цілісність інформації. Їх використання поширене у багатьох галузях, де потрібен високий рівень безпеки, особливо там, де учасники обміну даними не мають попередньої можливості встановити спільний секретний ключ.

Захищені мережеві з'єднання. Однією з найбільш поширених сфер застосування є протоколи безпеки, такі як TLS (Transport Layer Security), що забезпечують захищений обмін даними через Інтернет. Асиметричні алгоритми використовуються для встановлення з'єднання між клієнтом і сервером. Наприклад, сервер передає свій відкритий ключ клієнту, щоб клієнт міг зашифрувати дані. Лише сервер, маючи відповідний приватний ключ, зможе розшифрувати їх. Це дозволяє створити безпечний канал, через який можна обмінюватися сесійними ключами для подальшого симетричного шифрування.

Цифрові підписи. Асиметричні алгоритми використовуються для створення та перевірки цифрових підписів. Ця технологія гарантує, що повідомлення або документ було створено визначеним відправником і не змінено під час передачі. Наприклад, у фінансових транзакціях цифровий підпис підтверджує автентичність ініціатора платежу. У правовій сфері цифрові підписи використовуються для юридичних документів, забезпечуючи їхню цілісність і достовірність.

Обмін ключами. Асиметричні алгоритми забезпечують безпечний обмін ключами, що використовуються для симетричного шифрування. Наприклад, алгоритм Diffie-Hellman дозволяє двом сторонам створити спільний секретний ключ, навіть якщо вони обмінюються інформацією через незахищений канал. Це є критично важливим для VPN-з'єднань, мобільних додатків і хмарних сервісів.

Електронна комерція. В онлайн-магазинах та платіжних системах

асиметрична криптографія гарантує захист даних покупців, таких як номери банківських карт або особисті дані. Вона також забезпечує довіру між учасниками, підтверджуючи автентичність веб-сайтів за допомогою цифрових сертифікатів.

Блокчейн і криптовалюти. У блокчейн-технологіях асиметричні алгоритми використовуються для управління цифровими гаманцями. Приватний ключ дозволяє користувачам підписувати транзакції, тоді як відкритий ключ використовується для їхньої перевірки іншими учасниками мережі. Це забезпечує безпеку та прозорість операцій.

Автентифікація користувачів. У системах із високими вимогами до безпеки асиметрична криптографія забезпечує автентифікацію користувачів без передачі паролів. Наприклад, у SSH протоколі для входу на сервер використовуються ключі, що гарантує захищений доступ.

Хмарні сервіси. Хмарні провайдери використовують асиметричні методи для захисту переданих даних, управління доступом і шифрування збережених файлів. Відкритий ключ шифрує дані, які можуть бути дешифровані лише за допомогою відповідного приватного ключа.

Електронна пошта та системи обміну повідомленнями. У сервісах, таких як PGP (Pretty Good Privacy), асиметричні алгоритми використовуються для шифрування електронної пошти. Відправник шифрує повідомлення за допомогою відкритого ключа отримувача, забезпечуючи, що лише отримувач може його прочитати.

2.2.2.4 Сучасні алгоритми асиметричного шифрування.

2.2.2.4.1 RSA.

RSA (Rivest-Shamir-Adleman) — це асиметричний алгоритм криптографії, який використовується для шифрування, цифрових підписів і обміну ключами. Його безпека базується на складності факторизації великих чисел. RSA використовує пару ключів: відкритий для шифрування даних або перевірки підписів і приватний для дешифрування або створення підписів.

Підтримує різні розміри ключів (1024, 2048, 4096 біт), забезпечуючи високий рівень захисту.

Принцип роботи.

Принцип роботи RSA можна розділити на три основні етапи: генерацію ключів, шифрування та дешифрування.

На етапі генерації ключів обираються два великі прості числа p і q. Їхній добуток називається модулем, який є основою для ключів.

$$n = p \times x \tag{7}$$

Обчислюється значення функції Ейлера φ

$$\varphi(n) = (p-1)(q-1) \tag{8}$$

Потім обирається число ε (публічна експонента), яке є взаємно простим із $\varphi(n)$, і обчислюється d (приватна експонента), яка задовольняє умову:

$$\varepsilon \times d \equiv 1(mod\ \varphi(n)) \tag{9}$$

Для шифрування відправник використовує відкритий ключ та обчислює шифротекст:

$$\mathrm{C} = M^{\mathrm{e}}\ mod\ n \tag{10}$$

де M - це повідомлення у числовій формі.

Дешифрування виконується за допомогою приватного ключа:

$$M = C^{d}\ mod\ n \tag{11}$$

RSA також використовується для створення цифрових підписів. Відправник створює підпис, шифруючи хеш повідомлення своїм приватним ключем, а отримувач перевіряє його за допомогою відкритого ключа. Завдяки цьому RSA забезпечує конфіденційність, автентифікацію та цілісність даних.

Сфери застосування.

RSA використовується в багатьох сферах завдяки своїй здатності забезпечувати захист даних, автентифікацію та конфіденційність. Однією з основних сфер є захищені мережеві з'єднання, де RSA застосовується у протоколах TLS/SSL для створення безпечного каналу передачі даних між клієнтом і сервером. Це особливо важливо для веб-сайтів, онлайн-банкінгу та електронної комерції.

У цифрових сертифікатах RSA забезпечує автентифікацію серверів і користувачів, використовуючи цифрові підписи для підтвердження їхньої легітимності. Також алгоритм широко використовується для обміну ключами, де шифрує симетричні ключі, необхідні для подальшого шифрування даних.

RSA знаходить застосування в електронній пошті та системах обміну повідомленнями, таких як PGP, для шифрування та підписання повідомлень. У хмарних сервісах алгоритм використовується для захисту збережених і переданих даних. Також RSA важливий у блокчейн-технологіях, забезпечуючи автентифікацію транзакцій.

Переваги:

а) висока безпека: базується на складності факторизації великих чисел, що забезпечує стійкість до сучасних атак;

б) універсальність: використовується для шифрування, обміну ключами та цифрових підписів;

в) широка підтримка: інтегрований у більшість протоколів безпеки (TLS/SSL).

Недоліки:

а) обчислювальна складність: вимагає значних ресурсів, особливо при роботі з довгими ключами;

б) чутливість до квантових атак: квантові комп’ютери можуть зламати RSA за допомогою алгоритму Шора;

в) довгі ключі: для підвищення безпеки потрібні ключі великого розміру (2048 біт і більше), що впливає на продуктивність.

2.2.2.4.2 ECDSA.

ECDSA (Elliptic Curve Digital Signature Algorithm) — це криптографічний алгоритм для створення цифрових підписів, заснований на еліптичних кривих. Його основна перевага — високий рівень безпеки при використанні коротших ключів порівняно з іншими алгоритмами, такими як RSA. ECDSA працює на основі математичних властивостей точок на еліптичній кривій, що робить його стійким до багатьох типів атак. Алгоритм використовується для підтвердження автентичності даних, забезпечуючи цілісність і достовірність повідомлень. ECDSA широко застосовується у протоколах TLS, у блокчейн-технологіях для підписання транзакцій і у мобільних додатках завдяки своїй ефективності.

Принцип роботи.

Основою є математичне обчислення точок на кривій, яке забезпечує високу криптографічну стійкість навіть за умови використання коротких ключів.

Процес роботи ECDSA складається з двох основних етапів: створення підпису та його перевірки. Для створення підпису спочатку обчислюється хеш повідомлення, що підтверджує його цілісність. Потім використовується приватний ключ для створення двох чисел (r і s), які формують цифровий підпис. Цей процес базується на генерації випадкового числа k і використанні

еліптичних кривих для математичних операцій.

Для перевірки підпису застосовується відкритий ключ підписанта. Отримувач обчислює інші два значення (u1 і u2) на основі хешу повідомлення, r, s і відкритого ключа. Потім проводиться перевірка шляхом обчислення точки на кривій, що має збігатися з r. Якщо значення співпадають, підпис вважається дійсним.

ECDSA є удосконаленою версією DSA (Digital Signature Algorithm), але замість великих чисел використовує еліптичні криві. Це дозволяє досягти аналогічного рівня безпеки при значно коротших ключах, що робить ECDSA швидшим і ефективнішим у ресурсозалежних середовищах, таких як мобільні пристрої чи IoT.

Сфери застосування.

ECDSA широко застосовується в різних сферах завдяки своїй ефективності, високій безпеці та здатності працювати з короткими ключами. У протоколах безпеки, таких як TLS/SSL, ECDSA забезпечує автентифікацію серверів і клієнтів, а також створення цифрових сертифікатів для захищених з’єднань. Це важливо для забезпечення безпеки в електронній комерції, онлайн-банкінгу та хмарних сервісах.

У блокчейн-технологіях ECDSA є ключовим компонентом, що використовується для підписання транзакцій і перевірки їхньої автентичності. Алгоритм забезпечує захист цифрових гаманців, гарантує, що транзакції були ініційовані відповідним користувачем, і підтримує цілісність мережі.

У мобільних додатках і пристроях IoT ECDSA є ідеальним вибором завдяки коротким ключам, які зменшують навантаження на обчислювальні ресурси. Це дозволяє інтегрувати алгоритм у системи автентифікації та безпечного обміну даними.

ECDSA також використовується у системах електронного підпису документів, забезпечуючи їхню юридичну силу та автентичність, а також у криптографії електронної пошти, що гарантує безпеку повідомлень.

Переваги:

а) компактність ключів: забезпечує високий рівень безпеки при значно коротших ключах порівняно з RSA, що зменшує вимоги до пам’яті;

б) ефективність: виконує підписання та перевірку швидше, особливо на пристроях із обмеженими ресурсами, таких як IoT або мобільні пристрої;

в) безпечність: б азується на складності задачі дискретного логарифма на еліптичних кривих, що стійке до сучасних атак.

Недоліки:

а) складність реалізації: вимагає високої точності при налаштуванні параметрів кривої; помилки можуть призвести до вразливостей;

б) вразливість до квантових атак: квантові комп’ютери можуть зламати ECDSA за допомогою алгоритму Шора;

в) обчислювальна складність: хоча ключі коротші, операції на еліптичних кривих є складними, що може бути повільнішим у деяких сценаріях.

2.2.2.4.3 ECDH.

ECDH (Elliptic Curve Diffie-Hellman) — це асиметричний алгоритм для безпечного обміну ключами, заснований на еліптичних кривих. Його мета — дозволити двом сторонам створити спільний секретний ключ через незахищений канал зв’язку, не розкриваючи приватні ключі. Кожна сторона генерує пару ключів (публічний і приватний) і обмінюється публічними ключами. Спільний секрет обчислюється шляхом виконання математичних операцій на еліптичних кривих із використанням приватного ключа однієї сторони та публічного ключа іншої. ECDH забезпечує високу безпеку завдяки коротким ключам і широко використовується у VPN, TLS та мобільних додатках.

Принцип роботи.

Основою алгоритму є математична операція, що забезпечує, що обчислення спільного секрету можливе тільки зі знанням приватного ключа.

Кожна сторона (наприклад, клієнт і сервер) генерує пару ключів: приватний private_key та публічний public_key:

$$public_{key} = private_{key} \times G \tag{12}$$

де G — це базова точка на еліптичній кривій.

Сторони обмінюються своїми публічними ключами через незахищений канал. Потім кожна сторона обчислює спільний секретний ключ S шляхом множення отриманого публічного ключа іншої сторони на свій приватний ключ:

$$S = A_{private_{key}} \times B_{public_{key}} = B_{private_{key}} \times A_{private_{key}} \tag{13}$$

Ця властивість симетрії гарантує, що обидві сторони отримають однаковий спільний секрет, який може бути використаний як основа для подальшого симетричного шифрування.

ECDH є удосконаленням класичного Diffie-Hellman, який працює з великими простими числами. Завдяки використанню еліптичних кривих ECDH забезпечує той самий рівень безпеки при значно коротших ключах. Це робить його швидшим, менш ресурсоємним і більш придатним для мобільних пристроїв та IoT.

Сфери застосування.

ECDH широко застосовується в сучасних системах безпеки завдяки своїй здатності забезпечувати ефективний і безпечний обмін ключами через незахищені канали. У протоколах TLS/SSL ECDH використовується для створення спільного секретного ключа, який застосовується для симетричного

шифрування даних між клієнтом і сервером. Це є основою захищених з'єднань, таких як HTTPS, онлайн-банкінг та електронна комерція.

У VPN-системах ECDH дозволяє створити захищений тунель між клієнтом і сервером, де спільний ключ використовується для шифрування переданого трафіку. У мобільних додатках і пристроях IoT ECDH використовується завдяки своїй ефективності, яка важлива для ресурсозалежних середовищ.

Також алгоритм застосовується в системах електронної пошти та обміну повідомленнями, наприклад, у Signal Protocol, для захищеного обміну повідомленнями. Крім того, ECDH є важливим компонентом у хмарних сервісах, забезпечуючи безпечний доступ до даних і захищений обмін інформацією між пристроями. Його використання гарантує конфіденційність і безпеку даних у сучасних цифрових системах.

Переваги:

а) висока безпека: базується на складності дискретного логарифма на еліптичних кривих, що забезпечує стійкість до сучасних атак;

б) ефективність: забезпечує високий рівень безпеки з коротшими ключами, що знижує вимоги до обчислювальних ресурсів;

в) швидкість: підходить для ресурсозалежних середовищ, таких як IoT та мобільні додатки.

Недоліки:

а) складність реалізації: вимагає ретельного налаштування параметрів кривих; помилки можуть створити вразливості;

б) вразливість до квантових атак: як і більшість сучасних алгоритмів, може бути зламаний квантовими комп'ютерами.

2.3 Дослідження та аналіз пост-квантових криптографічних методів.

Пост-квантові методи криптографії — це новий клас криптографічних алгоритмів, розроблений для протистояння загрозам, які створюють квантові комп'ютери. Квантові обчислення здатні зламувати багато класичних криптографічних методів, таких як RSA, ECC (еліптичні криві) та алгоритми, засновані на задачі дискретного логарифма або факторизації великих чисел.

Пост-квантова криптографія забезпечує безпеку даних навіть у випадку появи потужних квантових комп'ютерів, які можуть виконувати атакувальні алгоритми, такі як алгоритм Шора.

Основна відмінність між класичними та пост-квантовими методами полягає в їхній математичній основі. Пост-квантові алгоритми базуються на задачах, які залишаються складними для квантових обчислень, таких як:

Решіткові задачі (Lattice-based cryptography). Використовуються проблеми, пов'язані з геометрією решіток у багатовимірному просторі, наприклад, проблема найближчого вектора (NVP).

Кодові задачі (Code-based cryptography). Базуються на складності декодування випадкових лінійних кодів.

Мультіваріантна криптографія (Multivariate quadratic equations). Використовують розв'язування систем багатоваріантних квадратичних рівнянь, що залишається складним завданням навіть для квантових комп'ютерів.

Задачі ізотопії ключів (Isogeny-based cryptography). Залучають еліптичні криві та задачі знаходження ізогеній між ними, що забезпечує високий рівень захисту.

Хеш-підписні алгоритми (Hash-based signatures). Ґрунтуються на криптографічних хеш-функціях і забезпечують довготривалу безпеку підписів.

2.3.1 Пост-квантовий аналог сучасних симетричних методів криптографії.

Кращий пост-квантовий аналог сучасних симетричних методів – це AES (Advanced Encryption Standard) з розширеними ключами, такими як AES-256.

Цей алгоритм не вимагає фундаментальної перебудови, оскільки їхній рівень безпеки базується на простих математичних операціях, таких як підстановки, перестановки та XOR, які залишаються складними для обчислення навіть для квантових комп'ютерів.

Проте основною проблемою стають атаки типу Grover's Algorithm, який може зменшити ефективну стійкість симетричних алгоритмів удвічі. Наприклад, AES-128 у квантовому середовищі забезпечує ефективну стійкість лише на 64 біти, що недостатньо для сучасних вимог безпеки. У зв'язку з цим використання AES-256 із 256-бітним ключем є найбільш перспективним пост-квантовим підходом до симетричного шифрування.

Пост-квантові симетричні алгоритми, такі як SPHINCS+ або алгоритми хеш-підпису, використовують додаткові механізми для підвищення стійкості до атак, заснованих на швидкому обчисленні квантових комп'ютерів. Основна різниця полягає в тому, що сучасні симетричні алгоритми, як AES-256, не потребують повної перебудови, а лише збільшення розміру ключа. Пост-квантові алгоритми часто базуються на хеш-функціях або решіткових задачах, які є повністю стійкими до квантових атак, але мають вищу обчислювальну складність.

Таким чином, AES-256 залишається найкращим вибором для симетричного шифрування навіть у пост-квантовому світі, за умови дотримання достатнього розміру ключа для протидії атакам квантових комп'ютерів.

2.3.2 Пост-квантовий аналог сучасних асиметричних методів криптографії.

Одним із найбільш перспективних пост-квантових аналогів сучасних асиметричних методів криптографії є Kyber, алгоритм, заснований на задачах решіткової криптографії (Lattice-based cryptography). Він пропонує стійкість до квантових атак і є ефективним замінником для сучасних асиметричних методів, таких як RSA та ECDSA.

Kyber базується на математичних задачах решіткової криптографії,

зокрема на проблемах пошуку коротких векторів у багатовимірних решітках. Його основа — Learning With Errors (LWE) або Ring-LWE, що включає обчислення в кільцях із поліномами.

Процес шифрування і обміну ключами виглядає так:

а) користувачі генерують пару ключів: приватний ключ для дешифрування та публічний для шифрування;

б) відправник використовує публічний ключ отримувача для шифрування даних і передачі їх;

в) отримувач за допомогою приватного ключа розшифровує повідомлення та відновлює спільний секретний ключ, який може бути використаний для подальшого симетричного шифрування.

Kyber відзначається високою продуктивністю й ефективністю навіть на сучасних апаратних пристроях, що робить його придатним для інтеграції в протоколи, такі як TLS.

Головна різниця між Kyber і традиційними алгоритмами (RSA, ECDSA) полягає в їхній математичній основі. RSA і ECDSA базуються на задачах, які квантові комп’ютери можуть ефективно зламати за допомогою алгоритму Шора, таких як факторизація великих чисел або дискретний логарифм. Kyber, натомість, використовує задачі решіткової криптографії, які навіть квантові комп’ютери не можуть вирішити за прийнятний час.

Також Kyber демонструє кращу продуктивність у порівнянні з RSA, особливо для великих ключів. Наприклад, RSA потребує дуже довгих ключів (4096 біт і більше) для забезпечення адекватної безпеки, тоді як Kyber досягає високого рівня безпеки з набагато коротшими ключами, що зменшує навантаження на мережу й обчислювальні ресурси.

3. РЕАЛІЗАЦІЯ СИСТЕМИ З УРАХУВАННЯМ ПОТЕНЦІЙНОЇ НЕБЕЗПЕКИ З БОКУ КВАНТОВИХ КОМП'ЮТЕРІВ.

3.1 Ідея та функціонал системи.

Додаток представляє собою сучасну фінансову платформу, яка дозволяє користувачам створювати, налаштовувати та керувати віртуальними картками для особистих і бізнес-потреб. Основна ідея полягає в спрощенні управління фінансами, оптимізації витрат та інтеграції з різними платіжними сервісами, зберігаючи високий рівень безпеки та зручності для користувачів.

Ідея додатку орієнтована на створення універсального інструменту для управління фінансами, що поєднує зручність, безпеку та інтеграцію з сучасними платіжними системами. Завдяки своїй адаптивності до потреб різних категорій користувачів — від приватних осіб до бізнесу — платформа має потенціал стати лідером у сфері фінансових технологій. Вона забезпечує повний контроль над витратами, захист коштів і оптимізацію бюджетів у зручному цифровому форматі.

Додаток пропонує розширений набір функцій, які забезпечують ефективне управління віртуальними картками та фінансовими транзакціями. Основна функціональність додатку спрямована на створення віртуальних карток для різних потреб користувачів. Користувач може легко створити картку, вибравши її тип і налаштувавши параметри, такі як ліміти витрат, обмеження за категоріями використання та географічними регіонами. Ця гнучкість дозволяє налаштувати картки під особисті потреби або для бізнес-цілей, таких як рекламні кампанії чи оплата послуг підрядників.

Додаток забезпечує детальне відображення історії транзакцій у вигляді інтерактивного дашборда. Користувачі можуть переглядати витрати за обраний період, аналізувати їх за категоріями та отримувати звіти. Це дає змогу не тільки стежити за фінансовими операціями, а й оптимізувати бюджет. Крім того, додаток підтримує функцію автоматичного повідомлення про підозрілі операції, допомагаючи користувачам миттєво реагувати на можливі ризики.

Платформа також інтегрується з популярними платіжними системами,

такими як PayPal, Google Pay і Apple Pay, що дозволяє користувачам використовувати віртуальні картки для онлайн-покупок або переказів. Для зручності, додаток підтримує автоматизацію платежів, наприклад, регулярне поповнення балансу карток, щоб забезпечити безперебійне функціонування підписок або постійних витрат.

Безпека є одним із ключових аспектів функціональності додатку. Усі карткові дані маскуються, а транзакції шифруються, щоб захистити інформацію від несанкціонованого доступу. Для входу в додаток і підтвердження операцій використовується двофакторна аутентифікація, а у випадку підозрілої активності користувач може миттєво заморозити картку.

Для бізнес-користувачів додаток пропонує функції KYC (Know Your Customer), що забезпечують відповідність регуляторним вимогам. Верифіковані користувачі отримують доступ до розширених функцій, таких як підвищені ліміти витрат та можливість делегувати фінансові операції іншим користувачам.

У додатку також передбачені функції для сімейного бюджету. Користувач може створити окремі картки для кожного члена родини з індивідуальними лімітами витрат, що спрощує управління фінансами в рамках родини. Додатково, передбачені інструменти для обмеження доступу до певних видів покупок, таких як азартні ігри чи розваги.

Усі ці функції інтегровані в інтуїтивно зрозумілий інтерфейс, що робить управління фінансами доступним для користувачів із будь-яким рівнем технічної підготовки. Додаток поєднує простоту використання з потужним функціоналом, орієнтованим як на індивідуальних користувачів, так і на бізнес.

Унікальні переваги додатку:

Інтерактивність і простота використання.

Інтерфейс створений для максимальної зручності, із зрозумілою навігацією та інтуїтивним дизайном. Користувач може легко створювати нові

картки, переглядати транзакції та керувати налаштуваннями.

Персоналізація.

Додаток дозволяє налаштовувати картки під індивідуальні потреби, включаючи вибір категорії витрат, інтеграцію з конкретними сервісами й навіть встановлення тимчасових обмежень.

Захист від шахрайства.

Високий рівень безпеки забезпечується сучасними криптографічними методами шифрування, а також функціями, такими як автоматичне блокування підозрілих операцій.

Можливості для бізнесу.

Платформа підтримує багатокористувацькі облікові записи, що дає можливість компаніям делегувати фінансові операції співробітникам без ризику перевищення витрат.

3.2 Розробка та опис функціональної схеми системи.

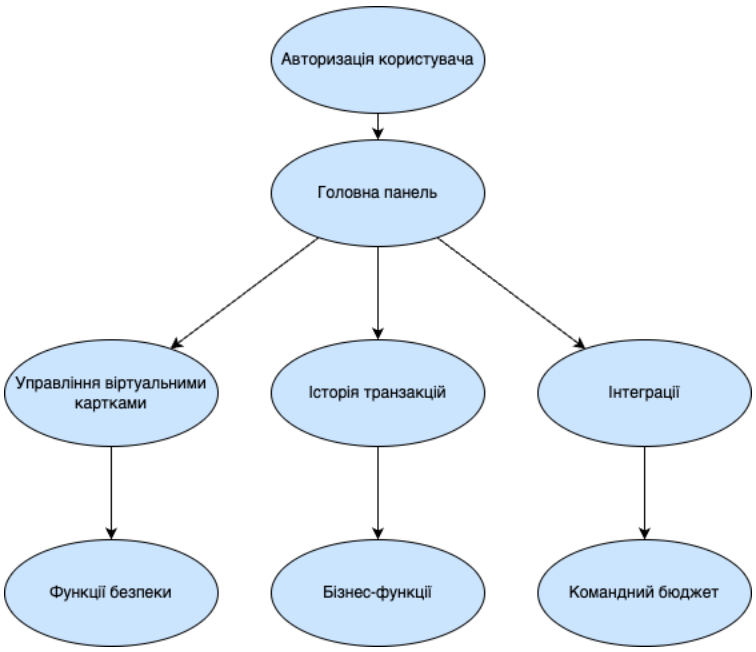


Рисунок 3.1 - Функціональна схема системи.

Функціональна схема відображає структуру та основні модулі системи

управління фінансами:

1) **Авторизація користувача** — початковий етап, де користувач проходить процес входу до системи. Це забезпечує доступ до головної панелі та відповідних функцій.
2) **Головна панель** — центральний вузол, що об’єднує всі ключові модулі системи. Користувачі отримують доступ до інших функцій через цю панель.
3) **Управління віртуальними картками** — забезпечує функції створення, редагування, блокування та налаштування лімітів карток. Цей модуль безпосередньо пов’язаний із функціями безпеки для захисту даних.
4) **Історія транзакцій** — надає користувачам можливість переглядати детальну інформацію про всі фінансові операції. Інтегрована з бізнес-функціями для аналізу та звітності.
5) **Інтеграції** — дозволяє підключати зовнішні платіжні системи або сервіси для розширення функціональності. Наприклад, інтеграція з PayPal або Google Pay.
6) **Командний бюджет** — модуль для управління фінансами команди, включаючи розподіл коштів і контроль витрат.

3.2.1 Алгоритм отримання транзакцій.

Рисунок 3.2 – Функціональна схема алгоритму отримання транзакцій.

Функціональна схема алгоритму описує процес обробки запиту на отримання транзакцій у додатку. Вона демонструє ключові етапи, починаючи від отримання запиту до відправки відповіді користувачеві.

Процес розпочинається з отримання API-запиту, коли користувач надсилає запит через фронтенд або інший клієнтський інтерфейс. Після цього виконується етап валідації даних, де перевіряється коректність отриманої інформації, зокрема, чи всі необхідні поля присутні, чи відповідають вони очікуваним форматам. Якщо дані не проходять валідацію, запит відхиляється.

Далі відбувається форматування фільтрів для підготовки запиту до бази даних. Цей етап включає обробку параметрів, таких як дати, суми або інші

критерії, що використовуються для пошуку транзакцій.

На етапі перевірки дозволів система оцінює, чи має користувач права доступу до запитуваних даних. Якщо доступ дозволено, алгоритм переходить до запиту до бази даних, використовуючи попередньо підготовлені фільтри. Якщо доступ заборонено, користувач отримує повідомлення про відмову.

Після отримання даних із бази відбувається форматування результатів, щоб підготувати інформацію у зрозумілому та зручному для користувача форматі. Нарешті, відформатовані дані надсилаються користувачу у вигляді відповіді на запит.

Система забезпечує цілісність транзакційного процесу, запобігаючи несанкціонованому доступу та помилкам у даних. Алгоритм має модульну структуру, що спрощує масштабування та підтримку.

3.2.2 Алгоритм перевірки прав доступу.

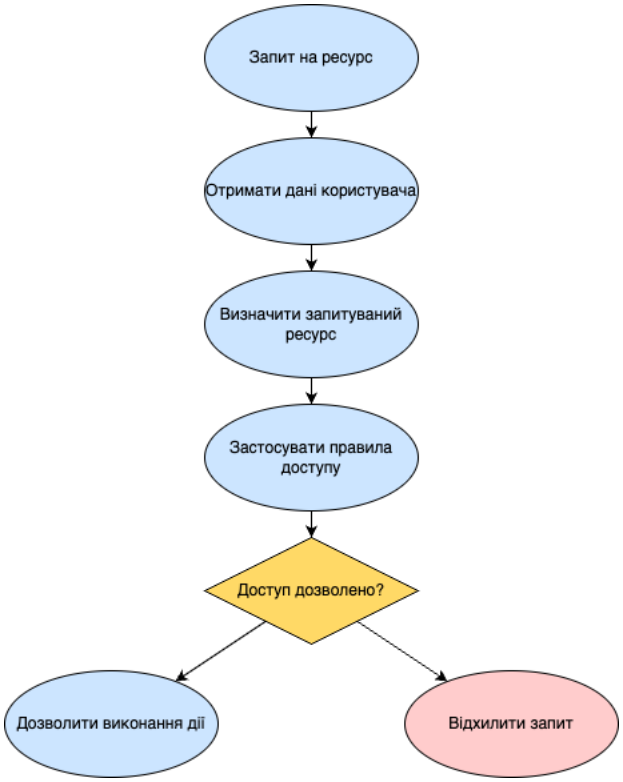


Рисунок 3.3 - Алгоритм перевірки прав доступу до ресурсу.

Процес перевірки дозволів у додатку забезпечує, щоб користувач мав

доступ лише до тих функцій і даних, які дозволені його роллю або статусом у системі. Цей процес є критично важливим для захисту конфіденційності та безпеки даних.

1) **Отримання контексту користувача**

Система отримує інформацію про користувача: його унікальний ідентифікатор, роль (наприклад, “адміністратор”, “користувач”, “менеджер команди”) та можливі додаткові метадані (наприклад, ID організації або команди).

2) **Визначення запитуваних ресурсів**

Під час запиту аналізується, до яких ресурсів або даних користувач намагається отримати доступ (наприклад, транзакції, фінансові звіти, налаштування карток).

3) **Застосування правил доступу**

У системі зберігаються правила доступу (Access Control Rules), які визначають, хто і що може робити. Правила можуть базуватися на ролях, ресурсах, типах дій (читання, запис, редагування) і додаткових обмеженнях (наприклад, лише у рамках певної команди).

4) **Перевірка співпадіння**

Система порівнює запит користувача з доступними йому правами. Якщо запит відповідає правилам доступу, дозволяється виконання дій.

5) **Результат перевірки**

Якщо доступ дозволено, система продовжує обробку запиту. У разі відмови користувач отримує повідомлення про помилку із зазначенням причин (наприклад, “недостатньо прав” або “заборонено роллю”).

3.2.3 Алгоритм перевірки транзакції

Алгоритм перевіряє, чи можна здійснити транзакцію, виконуючи кілька рівнів перевірок. Спочатку перевіряється статус картки, користувача та акаунта, щоб упевнитися, що вони активні. Потім здійснюється перевірка платіжних правил, визначаючи, чи дозволено або заборонено використовувати

певні атрибути транзакції. Далі оцінюється баланс акаунта, перевіряючи, чи вистачає коштів для проведення платежу. Останній етап включає перевірку лімітів витрат для користувача, картки та акаунта. Якщо будь-яка з перевірок не проходить, транзакція відхиляється, і її деталі реєструються для подальшого аналізу. У разі успішного виконання дозволяється продовження транзакції.

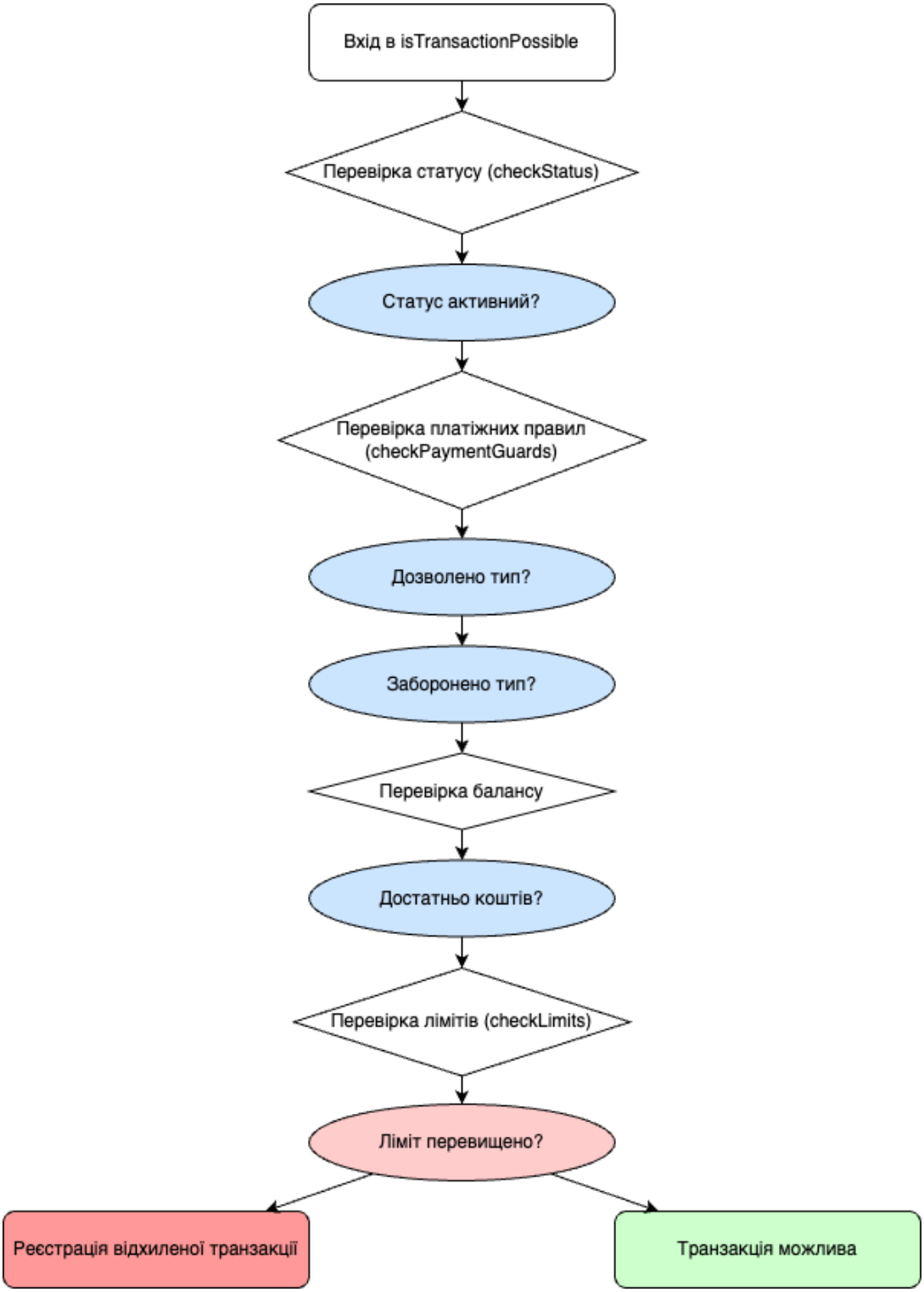


Рисунок 3.4 – Алгоритм перевірки можливості проведення транзакції.

3.3 Розробка та опис схем бази даних

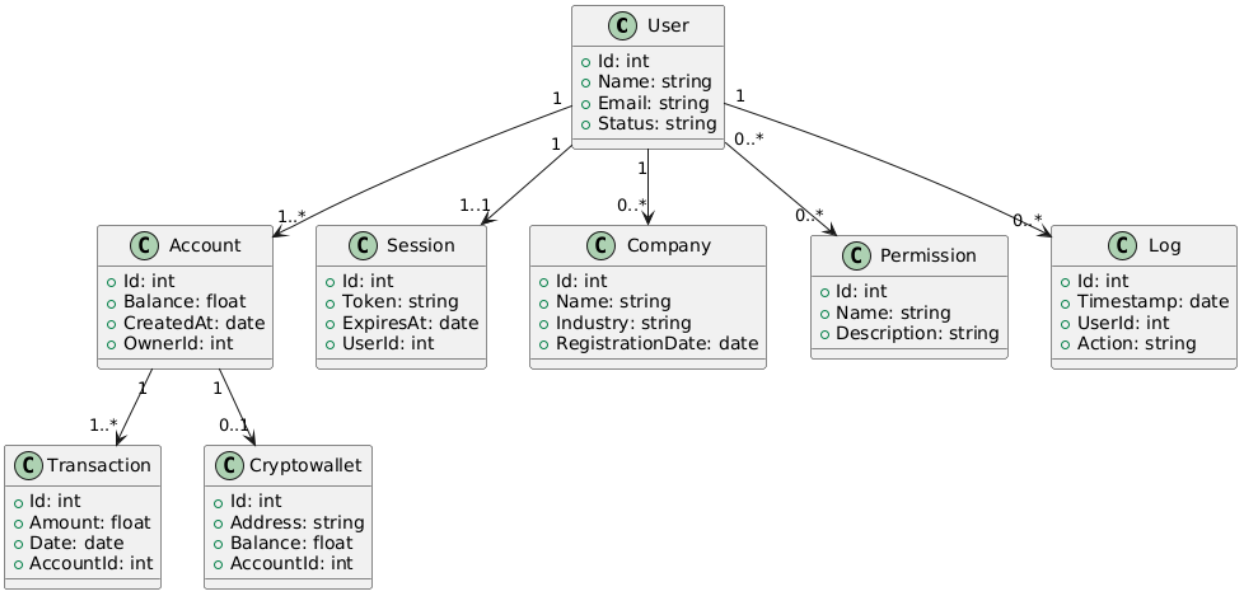


Рисунок 3.5 – UML-діаграма об’єктів бази даних системи

User.

Об’єкт User є ключовим компонентом системи, що відповідає за представлення користувачів. Він визначений у файлі User.schema.ts і містить основні атрибути, такі як ім’я, прізвище, роль, статус, а також додаткові поля для зберігання маркетингових UTM-міток та фінансових прогнозів. Цей об’єкт підтримує мітки часу, що дозволяє зберігати інформацію про створення і оновлення записів.

User має обов’язкові поля firstName і lastName, які ідентифікують користувачів. Атрибути Role і Status реалізовані через енумерації (UserRoleEnum і UserStatusEnum), що забезпечує чітке визначення ролей та статусів, необхідних для організації ієрархії доступу і управління правами користувачів.

Об’єкт також містить UTM-дані, такі як джерело трафіку, тип медіа та назву кампанії, що вказує на інтеграцію з маркетинговими системами. Додатково, компонент AccountForecasts, який є частиною об’єкта User, використовується для фінансового прогнозування, зокрема для оцінки обсягів

транзакцій і витрат.

У системі User є центральною одиницею, яка взаємодіє з багатьма іншими об’єктами. Він пов’язаний із Session для управління сеансами користувачів, із Account для асоціації фінансових рахунків, а також із логами дій, де фіксується активність користувачів. Об’єкт Referral, інтегрований із User, дозволяє реалізувати реферальну програму.

Роль User у системі полягає в ініціюванні основних операцій, таких як створення транзакцій, керування рахунками та взаємодія з іншими модулями. Завдяки атрибутам Role і Permission, User забезпечує контрольований і безпечний доступ до функціональних можливостей системи. Це робить об’єкт фундаментальним для реалізації бізнес-логіки, яка включає управління користувачами, їх фінансовими операціями і маркетинговою взаємодією.

Account.

Об’єкт Account в системі відповідає за представлення фінансових рахунків користувачів. Він визначений у файлі Account.schema.ts і є важливим елементом для зберігання інформації про баланс рахунку, дату його створення, а також асоціацію з конкретним користувачем через поле OwnerId. Об’єкт має властивість Balance, що відображає поточний фінансовий стан рахунку, і CreatedAt, яке дозволяє відслідковувати, коли саме рахунок було створено.

У проекті об’єкт Account виступає центральною точкою для виконання фінансових операцій. Він тісно пов’язаний із об’єктами Transaction, через які здійснюються транзакції, і Cryptowallet, що забезпечує інтеграцію з криптовалютними гаманцями. Цей зв’язок дозволяє системі обробляти як стандартні фінансові операції, так і транзакції, пов’язані з криптовалютами. Кожен рахунок може мати багато транзакцій, що вказує на його ключову роль у реєстрації і контролі всіх фінансових дій, пов’язаних із користувачем.

Роль Account також поширюється на управління фінансовими лімітами та комісіями, що реалізовано через інтеграцію з об’єктами Limit і Commission. Це дозволяє встановлювати обмеження на транзакції і автоматично обчислювати комісійні збори, пов’язані з рахунком. Крім того, рахунок може

бути пов’язаний із різними планами (Plan), що відкриває можливість для впровадження підписок або інших бізнес-моделей.

У загальному контексті проекту об’єкт Account виступає як базовий фінансовий модуль, через який користувачі здійснюють свої фінансові операції. Його інтеграція з іншими компонентами системи забезпечує гнучкість і функціональність у роботі з фінансами, що робить його критичним для реалізації ключових функцій системи.

Transaction.

Об’єкт Transaction у системі є ключовою сутністю, яка відповідає за обробку фінансових операцій. Він визначений у файлі Transaction.schema.ts і містить інформацію про суму транзакції, дату її виконання, а також посилання на рахунок, до якого вона належить, через поле AccountId. Транзакції можуть бути як вхідними, так і вихідними, що дозволяє відображати повний фінансовий потік, пов’язаний із конкретним рахунком.

Transaction виступає центральним елементом для фіксації будь-яких змін у фінансових даних. Взаємодія з об’єктом Account дозволяє точно реєструвати всі операції, що стосуються балансу рахунку. Це включає поповнення рахунку, списання коштів або інші фінансові дії. У системі транзакція також може бути пов’язана з криптовалютними операціями через інтеграцію з об’єктом Cryptowallet, що забезпечує підтримку транзакцій у різних валютах, включаючи криптовалюти.

У проекті об’єкт Transaction відіграє важливу роль у забезпеченні прозорості та відстежуваності фінансових операцій. Він може мати статуси, такі як “виконано”, “очікує підтвердження” або “відхилено”, що дозволяє відображати поточний стан операції. Цей статус може бути оброблений через асоціацію з відповідними типами або енумераціями.

Транзакції також інтегровані з системою лімітів та комісій. Об’єкт Transaction може враховувати фінансові обмеження, встановлені для рахунку, та автоматично розраховувати комісії за операцію через інтеграцію з об’єктом Commission. Це забезпечує дотримання бізнес-правил, пов’язаних із

фінансовими обмеженнями та витратами.

Крім основної функції обробки фінансових даних, Transaction є важливим джерелом даних для аналітики. Його записи можуть використовуватись для генерації звітів, відстеження фінансових показників або аналізу користувацької активності. Таким чином, Transaction є невід'ємною частиною фінансового ядра системи, забезпечуючи як технічну функціональність, так і підтримку бізнес-процесів.

Session.

Об'єкт Session є критичною частиною системи, що відповідає за управління сеансами користувачів. Він визначений у файлі Session.schema.ts і включає такі атрибути, як унікальний ідентифікатор сеансу, токен для аутентифікації, час його закінчення та посилання на користувача через UserId. Основна функція цього об'єкта полягає в забезпеченні зберігання та валідації активних сесій, що дозволяє підтримувати безпеку доступу до системи.

У системі об'єкт Session використовується для реалізації механізмів аутентифікації та авторизації. Коли користувач входить у систему, створюється новий запис у Session, який зберігає інформацію про токен і пов'язує його з конкретним користувачем. Це дозволяє системі перевіряти, чи є користувач авторизованим, і чи його сеанс є дійсним на момент виконання запиту. Термін дії токена визначається через атрибут ExpiresAt, що забезпечує автоматичне завершення сеансу після закінчення заданого часу.

Об'єкт Session також виступає важливим інструментом для управління безпекою в системі. Завдяки чіткій асоціації з об'єктом User, система може визначити, до яких ресурсів має доступ користувач, і реалізувати різнорівневу авторизацію. Це забезпечує контроль над доступом до чутливих даних і функцій.

У ширшому контексті об'єкт Session може використовуватись для моніторингу активності користувачів. Наприклад, записи сеансів можуть допомогти виявити підозрілу активність, таку як множинні невдалі спроби входу або використання одного токена з різних IP-адрес. Це додає рівень

захисту і сприяє виявленню потенційних загроз.

Роль Session у проекті полягає не лише у забезпеченні механізму аутентифікації, але й у створенні бази для відстеження та аналізу дій користувачів. Він є одним із ключових компонентів, що підтримують цілісність і безпеку системи, забезпечуючи водночас гнучкість у роботі з доступом та управлінням користувачами.

Cryptowallet.

Об’єкт Cryptowallet представляє криптовалютний гаманець і є важливою складовою фінансової частини системи. Він визначений у файлі Cryptowallet.schema.ts і використовується для управління криптовалютними активами користувачів. Серед його ключових атрибутів можна відзначити ідентифікатор гаманця, адресу, баланс та зв’язок із фінансовим рахунком через поле AccountId. Основна функція цього об’єкта полягає в інтеграції можливостей криптовалютних операцій із загальними фінансовими процесами системи.

У контексті проекту об’єкт Cryptowallet використовується для зберігання інформації про криптовалютні активи, що дозволяє користувачам зберігати, відправляти та отримувати криптовалюту. Завдяки атрибуту Address, кожен гаманець має унікальну адресу, яка використовується для взаємодії з блокчейном, що робить можливим виконання транзакцій у децентралізованій мережі.

Інтеграція з об’єктом Account дозволяє користувачам зручно керувати своїми фінансами, комбінуючи традиційні валюти та криптовалюту в одній системі. Це забезпечує універсальність та гнучкість у використанні системи, а також відкриває можливості для обробки транзакцій, включаючи обмін між фіатними і криптовалютними активами.

Об’єкт Cryptowallet також є основою для реалізації бізнес-функцій, таких як моніторинг балансу, управління ризиками та дотримання фінансових норм. Завдяки цьому система може автоматично відслідковувати рух коштів у криптовалютних гаманцях, проводити аналіз активності та забезпечувати

безпеку транзакцій.

Роль Cryptowallet у проекті виходить за рамки зберігання криптовалютних активів. Він є ключовим компонентом для інтеграції з блокчейн-технологіями, підтримки складних фінансових операцій і реалізації нових бізнес-моделей, які базуються на криптовалюті. Це робить Cryptowallet важливим елементом фінансової інфраструктури системи.

Permission.

Об’єкт Permission відіграє ключову роль у системі управління доступом. Він визначений у файлі Permission.schema.ts і використовується для опису прав, які надаються користувачам або їхнім ролям у системі. Цей об’єкт включає атрибути, що описують назву дозволу, його унікальність і асоційовані з ним елементи, такі як доступ до певних моделей, полів або точок API. Основна функція Permission полягає в забезпеченні детального контролю над тим, які операції можуть виконувати користувачі або групи.

У проекті Permission є базовим компонентом для реалізації авторизації. Коли користувач або роль намагається отримати доступ до певної функції або ресурсу, система перевіряє відповідні дозволи, щоб визначити, чи є такий доступ дозволеним. Це забезпечує захист чутливих даних і функціоналу від несанкціонованого використання.

Об’єкт Permission також підтримує гнучку конфігурацію доступу, що дозволяє адміністраторам системи налаштовувати права для окремих користувачів або груп залежно від їхньої ролі або завдань. Це робить можливим реалізацію принципу найменших привілеїв, коли користувач отримує лише ті права, які необхідні для виконання його функцій.

Додатково, Permission інтегрується з іншими об’єктами системи, такими як User і Role, утворюючи ієрархічну модель доступу. Це дозволяє легко розширювати систему, додаючи нові типи дозволів або змінюючи існуючі. У поєднанні з механізмами логування, система може зберігати записи про використання дозволів, що сприяє підвищенню безпеки та прозорості.

Роль об’єкта Permission полягає в тому, щоб забезпечити контрольований

доступ до ресурсів і функціоналу системи. Він є важливим інструментом для управління доступом, підтримки бізнес-логіки та захисту від потенційних загроз. Завдяки своїй універсальності і інтеграції з іншими модулями, Permission є критично важливим елементом у реалізації надійної системи авторизації.

3.4 Розробка системного дизайну.

Системний дизайн цього проекту можна описати як багаторівневу архітектуру з акцентом на модульність, безпеку, масштабованість і інтеграцію з фінансовими та криптовалютними сервісами. Ось основні аспекти дизайну:

Проект базується на трирівневій архітектурі, яка включає рівні презентації, бізнес-логіки та зберігання даних. На рівні презентації клієнтські додатки (веб або мобільні) взаємодіють із системою через API, реалізований на основі NestJS. API відповідає за маршрутизацію запитів і обробку даних перед передачею їх у бізнес-логіку.

Бізнес-логіка є серцем системи і відповідає за виконання всіх операцій, таких як обробка користувацьких запитів, управління сесіями, виконання транзакцій, обробка криптовалютних операцій і забезпечення доступу на основі ролей і дозволів. Логіка реалізована у вигляді сервісів, кожен з яких відповідає за певну область, наприклад, керування користувачами, рахунками, транзакціями, криптовалютними гаманцями та дозволами.

Рівень даних включає базу даних, яка використовується для зберігання інформації про користувачів, рахунки, транзакції, сесії, дозволи та інші об’єкти. Система використовує MongoDB, що забезпечує гнучкість і масштабованість завдяки її документно-орієнтованій моделі. Для роботи з базою даних використовується Mongoose, який забезпечує ORM-функціональність і дозволяє працювати зі схемами даних у вигляді моделей.

У системі є чіткий розподіл відповідальностей між основними об’єктами. User відповідає за зберігання інформації про користувачів, Account забезпечує облік фінансів, Transaction обробляє фінансові операції, а

Cryptowallet інтегрує можливості роботи з криптовалютами. Session відповідає за управління авторизацією, а Permission та ролі користувачів забезпечують доступ до функціоналу на основі чітко визначених прав.

Для забезпечення безпеки використовуються механізми аутентифікації та авторизації. Аутентифікація реалізована через токени доступу (JWT), що зберігаються у зв'язаному об'єкті Session. Авторизація реалізована через механізми ролей і дозволів, що визначають, які операції може виконувати користувач.

Проект інтегрує криптовалютні функції через об'єкт Cryptowallet, що дозволяє зберігати, отримувати та відправляти криптовалюту. Це робить систему гнучкою для роботи як із традиційними валютами, так і з криптовалютами.

Для підвищення безпеки та продуктивності система використовує журнали (об'єкт Log), які фіксують всі важливі події, наприклад, авторизаційні спроби, фінансові операції або зміни конфігурації. Це забезпечує можливість аудиту та моніторингу.

Система легко масштабується завдяки використанню контейнеризації (Docker Compose) і хмарних сервісів. Мікросервісна архітектура дозволяє виділити окремі модулі, наприклад, для роботи з транзакціями або криптовалютними операціями, у вигляді незалежних сервісів, що можуть бути розгорнуті окремо.

Основні компоненти системи розділені на кілька мікросервісів, кожен з яких відповідає за конкретну частину бізнес-логіки та функціональності.

Мікросервіс Application є центральним компонентом, що обробляє користувацькі запити і забезпечує взаємодію між іншими сервісами. Він відповідає за автентифікацію, управління сесіями, обробку дозволів, а також виконує роль API-шлюзу, через який клієнтські додатки взаємодіють із системою. Цей сервіс також координує роботу з об'єктами, такими як User, Account, і Transaction, забезпечуючи інтеграцію бізнес-логіки.

PaymentControl фокусується на обробці фінансових операцій. Він

відповідає за застосування бізнес-правил для транзакцій, включаючи облік лімітів і розрахунок комісій. Цей сервіс забезпечує прозорість і відповідність фінансових операцій правилам системи. Він тісно інтегрований із об’єктами Transaction та Account, виконуючи важливу роль у верифікації та обробці транзакцій.

WebhookService забезпечує інтеграцію з зовнішніми системами та сервісами. Його головна мета — отримання і обробка подій від сторонніх сервісів, таких як платіжні шлюзи або інструменти аналітики. Цей сервіс працює з вебхуками, дозволяючи системі реагувати на події в реальному часі, наприклад, оновлення статусу платежу або повідомлення про завершення операцій.

DB є централізованим сховищем даних, де зберігається вся необхідна інформація про користувачів, рахунки, транзакції, криптовалютні гаманці, сесії та дозволи. Цей сервіс базується на MongoDB і забезпечує високу доступність та швидкість обробки запитів завдяки документно-орієнтованій моделі. Він працює у взаємодії з іншими сервісами, забезпечуючи зберігання та доступ до даних.

Metabase виконує роль інструменту аналітики та візуалізації даних. Цей сервіс використовує інформацію з бази даних для створення звітів і дашбордів, що дозволяють адміністраторам і бізнес-аналітикам отримувати детальну інформацію про фінансові операції, користувацьку активність і ефективність системи. Завдяки Metabase користувачі можуть відстежувати ключові показники, аналізувати тенденції та приймати обґрунтовані рішення.

Цей дизайн забезпечує модульність, що спрощує підтримку та додавання нових функцій, гнучкість у роботі з різними фінансовими інструментами та безпеку даних завдяки чітко визначеним правилам доступу та механізмам журналювання.

3.5 Впровадження пост-квантових криптографічних алгоритмів шифрування.

В протоколі TLS 1.3, що зараз широко використовується в усіх передових платформах, використовуються ECDH (DH), або RSA для обміну ключами та формування спільного ключа для подальшого обміну інформацією за допомогою симетричного шифрування. Найнадійніший та найпоширюваніший сучасний симетричний алгоритм шифрування – це AES з довжиною ключа 128 біт (AES-128).

Для того, щоб зробити систему стійкою для атак за допомогою квантових обчислень, нам знадобляться пост-квантові криптографічні методи або сучасні методи, стійкі до квантових атак.

Так як AES відносно стійкий до квантових атак, але його ефективність значно зменшується через квантові обчислення, нам потрібно збільшити розмір ключа до 256 біт. Це забезпечить стійкий рівень захисту навіть в епоху квантових обчислень.

Більш вразливими до квантових атак є сучасні асиметричні криптографічні алгоритми, що використовуються для обміну ключами та цифрових підписів. Наприклад такі найрозповсюдливіші алгоритми, як ECDSA для цифрових підписів та (EC)DH(E) або RSA для обміну ключами.

Використовувати ці алгоритми в квантову епоху не можна. Але є вже пост-квантові спадкоємці для цих сучасних алгоритмів – це Kyber для обміну ключів та Dilithium для цифрових підписів.

Для впровадження цих нових методів пост-квантової криптографії нам треба:

1) **Встановлення серверу Nginx.**

Треба також переконатись, що наш Nginx-сервер підтримує використання OpenSSL. Якщо ні, треба перекомпілювати Nginx для роботи з бібліотекою OpenSSL.

2) **Встановлення OpenSSL з підтримкою пост-квантових алгоритмів.**

OpenSSL наразі інтегрує пост-квантові алгоритми, експериментуючи з ними через додаткові бібліотеки: OpenSSL від OQS (Open Quantum Safe), який підтримує Kyber, Dilithium та інші пост-квантові алгоритми.

```
git clone --branch openssl-1.1.1 https://github.com/open-quantum-safe/openssl.git
cd openssl
./config
make -j$(nproc)
sudo make install
```

Щоб перекомпілювати із підтримкою поточної версії OpenSSL:

```
./configure --with-http_ssl_module --with-openssl=/path/to/your/openssl
make -j$(nproc)
sudo make install
```

3) **Налаштування TLS 1.3 із використанням Kyber для обміну ключами.**

Згенеруємо приватний ключ із використанням Kyber:

```
oqs-openssl genpkey -algorithm kyber512 -out kyber_private.key
```

Згенеруємо сертифікатний запит (CSR):

```
oqs-openssl req -new -key kyber_private.key -out kyber_request.csr
```

Підпишимо CSR за допомогою Dilithium:

```
oqs-openssl x509 -req -in kyber_request.csr -signkey kyber_private.key -out kyber_certificate.pem -days 365
```

Налаштуємо Nginx для використання TLS 1.3 із Kyber:

```
server {
    listen 443 ssl http2;
    ssl_certificate /path/to/kyber_certificate.pem;
    ssl_certificate_key /path/to/kyber_private.key;
    ssl_protocols TLSv1.3;
    ssl_ciphers KYBER512:AES256-GCM-SHA384;
}
```

4) **Налаштування Nginx для використання AES-256.**

```
ssl_ciphers AES256-GCM-SHA384:KYBER512;
ssl_prefer_server_ciphers on;
```

5) **Генерування сертифікати за допомогою алгоритму Dilithium.**

Згенеруємо приватний ключ:

```
oqs-openssl genpkey -algorithm dilithium2 -out
dilithium_private.key
```

Згенеруємо CSR:

```
oqs-openssl req -new -key dilithium_private.key -out
dilithium_request.csr
```

Створимо самопідписаний сертифікат:

```
oqs-openssl x509 -req -in dilithium_request.csr -signkey
dilithium_private.key -out dilithium_certificate.pem -days 365
```

Вкажемо нові сертифікати в конфігурації Nginx:

```
ssl_certificate /path/to/dilithium_certificate.pem;
ssl_certificate_key /path/to/dilithium_private.key;
```

6) **Тестування з клієнтами, що підтримують пост-квантові алгоритми (QQS OpenSSL).**

Ці дії впроваджують сучасні пост-квантові криптографічні алгоритми в Nginx-сервер. Вони забезпечують захист від квантових атак на етапі обміну ключами, шифрування даних та підпису сертифікатів. Система стає стійкою до сучасних і майбутніх загроз, зберігаючи при цьому сумісність із традиційними клієнтами.

ВИСНОВКИ

У процесі виконання дипломної роботи було досліджено та реалізовано систему захисту даних з використанням постквантової криптографії, інтегрованої у веб-додаток фінансової тематики. Основною метою роботи було забезпечення надійності криптографічного захисту даних в умовах зростання обчислювальної потужності, зокрема з урахуванням можливостей квантових комп’ютерів.

У ході дослідження було здійснено глибокий аналіз сучасних методів криптографії, включаючи симетричні, асиметричні та хешувальні алгоритми. Серед них було виявлено ключові переваги та недоліки, що дозволило зробити висновки про їхню вразливість до квантових атак. Особлива увага приділялася алгоритмам AES, RSA, ECDSA, а також сучасним постквантовим методам, таким як Kyber і Dilithium. Було встановлено, що постквантові алгоритми мають потенціал забезпечити тривалий захист даних у разі широкого впровадження квантових обчислень.

Окрім теоретичного аналізу, в роботі було реалізовано веб-додаток, побудований за допомогою технологій Node.js, NestJS, та MongoDB. Система інтегрує функціонал управління фінансовими транзакціями, включаючи перевірки на ліміти витрат, управління статусами облікових записів та карток, а також модулі шифрування й безпеки. В основі архітектури лежать ретельно розроблені моделі, такі як Account, User, Card, Transaction та інші, які демонструють складну систему взаємозв’язків між сутностями.

Було створено UML-діаграми, що відображають структуру бази даних і алгоритмів, а також детальні функціональні схеми ключових алгоритмів, таких як перевірка лімітів, управління платежами та обробка транзакцій. Це дозволило не лише формалізувати взаємодію компонентів системи, але й виявити точки потенційного покращення продуктивності та безпеки.

Результатом роботи стало впровадження криптографічного протоколу з підтримкою Perfect Forward Secrecy (PFS), модифікованого для використання постквантового алгоритму Kyber. Такий підхід дозволив забезпечити

конфіденційність навіть у разі компрометації ключів, а також підготувати систему до появи квантових атак.

Розроблений веб-додаток демонструє широкий спектр функцій, таких як інтеграція управління транзакціями, обробка фінансових даних, підтримка багатокористувацького режиму та детальні механізми безпеки. Для цього було реалізовано перевірку дозволів доступу, захист сесій користувачів, а також модулі управління криптогаманцями та реферальними програмами.

Під час розробки системи було враховано сучасні вимоги до продуктивності та масштабованості, що робить її придатною для використання у фінансових та комерційних додатках. Використання таких технологій, як Mongoose для управління базою даних і NestJS для створення сервісів, забезпечило модульність і розширюваність архітектури.

Загалом, виконана робота доводить важливість впровадження постквантових методів у сучасні системи захисту даних. Розроблений додаток є прикладом практичного застосування таких методів у сфері фінансів, що відкриває можливості для подальшого розвитку подібних рішень. Перспективними напрямами для майбутніх досліджень можуть бути вдосконалення алгоритмів шифрування, розробка більш оптимальних архітектур для управління великими обсягами даних та інтеграція з іншими постквантовими протоколами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Katz, J., & Lindell, Y. *Introduction to Modern Cryptography* [Книга]. – 2020. – Режим доступу до ресурсу: https://www.crcpress.com.
2. Menezes, A. J., Vanstone, S. A., & Oorschot, P. C. *Handbook of Applied Cryptography* [Книга]. – 2001. – Режим доступу до ресурсу: https://www.crcpress.com.
3. Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C* [Книга]. – 2015. – Режим доступу до ресурсу: https://www.wiley.com.
4. Bernstein, D. J., & Lange, T. *Post-quantum cryptography – Dealing with the fallout of physics success* [Наукова стаття]. – 2017. – Режим доступу до ресурсу: https://eprint.iacr.org/2017/314.
5. Koblitz, N., & Menezes, A. J. *The brave new world of post-quantum cryptography* [Наукова стаття]. – 2016. – Режим доступу до ресурсу: https://doi.org/10.1016/j.future.2015.12.001.
6. Alagic, G., et al. *Status report on the second round of the NIST post-quantum cryptography standardization process* [Наукова стаття]. – 2022. – Режим доступу до ресурсу: https://doi.org/10.6028/NIST.IR.8309.
7. National Institute of Standards and Technology (NIST). *Post-Quantum Cryptography Project* [Електронний ресурс]. – Режим доступу до ресурсу: https://csrc.nist.gov/projects/post-quantum-cryptography.
8. Cloudflare. *What is Perfect Forward Secrecy (PFS)?* [Електронний ресурс]. – Режим доступу до ресурсу: https://www.cloudflare.com/learning/ssl/what-is-perfect-forward-secrecy/.
9. OpenSSL Documentation. *TLS/SSL and Cryptography* [Електронний ресурс]. – Режим доступу до ресурсу: https://www.openssl.org/docs/.
10. IBM Quantum. *Understanding quantum computing and cryptography* [Електронний ресурс]. – Режим доступу до ресурсу: https://www.ibm.com/quantum-computing/learn/what-is-quantum-computing.

11. OWASP Foundation. *Cryptography Cheat Sheet* [Електронний ресурс]. – Режим доступу до ресурсу: https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html.
12. Google Security Blog. *Rolling out TLS 1.3 and post-quantum cryptography* [Електронний ресурс]. – Режим доступу до ресурсу: https://security.googleblog.com/.
13. Microsoft Research. *Post-Quantum Cryptography and its applications* [Електронний ресурс]. – Режим доступу до ресурсу: https://www.microsoft.com/en-us/research/publication/.
14. W3Schools. *JavaScript Encryption Documentation* [Електронний ресурс]. – Режим доступу до ресурсу: https://www.w3schools.com/.
15. Mozilla Developer Network. *TLS Protocols and Implementation* [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/Security/TLS.

Додаток А – Код програм

Код – контроль-перевірка транзакції.

```
private async checkLimits(objects: any[], payload: { user:
UserDocument, data: any, startDate: Date, session:
ClientSession }) {
  const results = await Promise.all(objects.map(async (object: {
    target: MemberDocument | AccountDocument | CardDocument,
reason: RejectReasonEnum
  }) => {
    const model =
object.target.constructor.prototype.collection.modelName.toLower
Case();

    if (object.target.limitAmount !== undefined &&
object.target.limitPeriod && object.target.limitPeriod !==
LimitPeriodEnum.Unlimited) {
      const spent = await calcSpentMongoDB(this[model +
"Model"], object.target._id.toString(), payload.session);
      if (spent + payload.data.amountValue >
object.target.limitAmount) {
        return { result: false, reason: object.reason };
      }

      return { result: true };
    }
    return { result: true };
  }));

  return results.filter(res => res.result === false);
}

private async checkPaymentGuards(data: {
  data: any, card: CardDocument, account: AccountDocument,
member: MemberDocument, user: UserDocument, startDate: Date
}) {
  const unsuitable = [];
  const paymentGuards = await this.paymentGuardModel.find();

  for (const paymentGuard of paymentGuards) {
    if (paymentGuard.cardType !== undefined &&
paymentGuard.cardType !== data.card.type.name) continue;

    if (paymentGuard?.allowed?.length
&& !paymentGuard.allowed.includes(data.data[paymentGuard.field])
)
      unsuitable.push({ result: false, reason:
RejectReasonEnum.NotAllowedMerchant });

    if (paymentGuard?.prohibited?.length &&
paymentGuard.prohibited.includes(data.data[paymentGuard.field]))
      unsuitable.push({ result: false, reason:
```

```
RejectReasonEnum.ProhibitedMerchant });
  }
  return unsuitable;
}

private checkStatus(account: AccountDocument, member:
MemberDocument, card: CardDocument) {
  const unsuitable = [];
  if (card.status !== CardStatusEnum.Enabled)
unsuitable.push({ result: false, reason:
RejectReasonEnum.CardNotActive });
  if (member.status !== MemberStatusEnum.Enabled)
unsuitable.push({ result: false, reason:
RejectReasonEnum.MemberNotActive });
  if (account.status !== AccountStatusEnum.Enabled)
unsuitable.push({ result: false, reason:
RejectReasonEnum.AccountNotActive });

  return unsuitable;
}

private async isTransactionPossible(payload: {
  data: any, card: CardDocument, account: AccountDocument,
member: MemberDocument, user: UserDocument, startDate: Date,
session: ClientSession
}) {
  const unsuitable = [];
  unsuitable.push(...this.checkStatus(payload.account,
payload.member, payload.card));

  unsuitable.push(...await this.checkPaymentGuards(payload));

  if (payload.data.amountValue > payload.account.balance) {
    unsuitable.push({ result: false, reason:
RejectReasonEnum.NotEnoughBalance });
  }

  unsuitable.push(...await this.checkLimits(
    [ { target: payload.member, reason:
RejectReasonEnum.MemberLimit }, { target: payload.card, reason:
RejectReasonEnum.CardLimit }, { target: payload.account, reason:
RejectReasonEnum.AccountLimit } ],
    payload
  ));

  if (unsuitable.length) {
    await this.addRejectedTransaction({ ...payload, reasons:
unsuitable.map(result => result.reason) });
    throw new ForbiddenException();
  }
}
```

Код – перевірка прав доступу користувача до ресурсу бази даних.

```
export const lookupMemberDataQuery = (query: any[], model:
string, user: any) => {
  query.push(...[
    { $lookup: {
      from: "members",
      let: { accountId: model === "account" ? "$_id" :
"$account", userId: user.sub ? new
mongoose.Types.ObjectId(user.sub) : null },
      pipeline: [
        { $match: {
          $expr: {
            $and: [
              { $eq: [ "$account", "$$accountId" ] },
              { $eq: [ "$user", "$$userId" ] },
              { $in: [ "$status", [MemberStatusEnum.Enabled,
MemberStatusEnum.Suspended] ] }
            ]
          }
        } }
      ],
      as: "memberData"
    } },
    { $unwind: {
      path: `$memberData`,
      preserveNullAndEmptyArrays: true
    } }
  ]);
}
export function lookupPermissions(query: any[]) {
  query.push(...[ // lookup member permissions and all
permissions
    { $lookup: {
      from: "permissions",
      localField: "memberData.permissions",
      foreignField: "_id",
      as: "memberPermissions"
    } },
    { $lookup: {
      from: "permissions",
      as: "allPermissions",
      pipeline: [
        { $addFields: { justForSyntax: true } }
      ]
    } }
  ]);
}
export function addPermissionFilter(query: any[], model: string,
user: any) {
  if (!user.roles.includes(UserRoleEnum.Admin)
&& !user.roles.includes(UserRoleEnum.Support)) { // if regular
user, then filter by permissions
```

```
    if (model !== "account") {
      query.push({ $match: { // filtering by permissions
        $or: [
          { memberPermissions: { // or if at least one
permission of member permissions contains required model then
let go further
            $elemMatch: {
              models: {
                $elemMatch: { $eq: model }
              }
            }
          } },
          { $expr: { // or if none of the existing permissions
contains the requested model
            $eq: [
              { $size: {
                  $filter: {
                    input: "$allPermissions",
                    as: "item",
                    cond: {
                      $eq: [
                        { $size: {
                            $filter: {
                              input: "$$item.models",
                              as: "model",
                              cond: { $eq: ["$$model",
model] } }
                          } },
                        0
                      ]
                    }
                  }
                } },
              { $size: "$allPermissions" }
            ]
          } },
          { $expr: { // or if you are the owner of the current
document
            $eq: [ "$memberData._id", "$member" ]
          } }
        ]
      } });
    }
  }

}

export function filterByPermissions(query: any[], model: string,
user) {
  lookupMemberDataQuery(query, model, user);

  lookupPermissions(query);
```

```
  addPermissionFilter(query, model, user);

  if (!user.roles.includes(UserRoleEnum.Admin)
&& !user.roles.includes(UserRoleEnum.Support)) { // exception
with admin and fetching accounts
    query.push({ $match: { memberData: { $exists: true } } });
  }

  filterFieldsByPermissions(query, model);

  const removeMemberData: any = {};
  if (model !== "account") {
    removeMemberData.memberData = 0;
  }

  query.push({ $project: { memberPermissions:
0, ...removeMemberData } })
}
```

Код – випуск віртуальної картки.

```
async linkCards(userId: string, dto: LinkCardDto) {
  const cardType = await
this.cardTypeModel.findById(dto.cardType);
  if (!cardType) throw new HttpException("CardType not found",
HttpStatus.NOT_FOUND);
  if (cardType.status !== CardTypeStatusEnum.Active) throw new
HttpException("CardType is not active", HttpStatus.BAD_REQUEST);

  const member = await this.memberModel.findOne({ account:
dto.accountId, user: userId })
    .select("+user")
    .populate({
      path: "user",
      select: "+email +telegram"
    });
  if (!member) throw new HttpException("Member not found",
HttpStatus.NOT_FOUND);
  if (member.status !== MemberStatusEnum.Enabled) throw new
HttpException("Member not enabled", HttpStatus.FORBIDDEN);

  return transaction(this.connection, async session => {
    const account: any = await
this.accountModel.findById(dto.accountId)
      .select("+balance +commissions")
      .populate("commissions")
      .populate("referrerCode")
      .populate({
        path: "legalRepresentative",
        populate: { path: "user" }
      })
      .session(session);
```

```
    if (cardType.accessLevel >
account.legalRepresentative.user.accessLevel) {
      console.log(cardType.accessLevel,
account.legalRepresentative.user.accessLevel)
      cardType.status = CardTypeStatusEnum.NoAccess;
    }
    if (cardType.status !== CardTypeStatusEnum.Active) {
      throw new HttpException("This type of card is not
available. Please contact to support for information",
HttpStatus.BAD_REQUEST);
    }

    const commissions = account.commissions.filter(commission =>
commission._id.toString() === cardType.commission.toString());
    const payoutPayments: any = [], feePayments: any = [];
    const totalCommissionSum =
generatePayoutsByCommission(account, this.paymentModel,
commissions[0], dto.amount, payoutPayments, feePayments, 0,
account.freeCards || 0);

    if (account.balance < totalCommissionSum) throw new
HttpException("Insufficient funds on balance",
HttpStatus.FORBIDDEN);

    const cards = await this.cardModel.aggregate([
      { $match: {
        type: cardType._id,
        status: CardStatusEnum.Pending,
      } },
      { $lookup: {
        as: "company",
        foreignField: "_id",
        localField: "company",
        from: "companies"
      } },
      { $match: {
        "company.status": CompanyStatusEnum.Enabled
      } },
      { $limit: dto.amount }
    ], { session });

    let availableCards;

    if (cards.length < dto.amount) {
      availableCards = (await
this.getAvailableCardsByTypes(session)).str;

      await this.mailingService.sendTelegramNotification(
        "-852078196",
        `ZeroCard NOT ENOUGH CARDS AVAILABLE: \n\nAmount:
${dto.amount}\nType: ${cardType.name}\nAccountId: ${account._id}
\n\nAvailable cards: \n${availableCards}`
      );
```

```
      throw new HttpException("Issue of cards is temporarily
unavailable. Try again later.", HttpStatus.BAD_REQUEST);
    }

    const cardPayload: any = {
      status: CardStatusEnum.Enabled,
      user: userId,
      member,
      account,
      name: dto.name,
      limitPeriod: dto.limitPeriod || LimitPeriodEnum.Unlimited,
      consentUrl: null,
      linkedAt: new Date()
    }
    if (dto.limit !== undefined) cardPayload.limitAmount =
dto.limit;

    await this.cardModel.updateMany(
      { _id: { $in: cards.map(card => card._id) } },
      { $set: cardPayload },
      { session }
    );

    availableCards = (await
this.getAvailableCardsByTypes(session)).str;

    await Promise.all(payoutPayments.map(async payout => {
      await payout.save({ session });
      await
this.databaseService.reCalcBalanceMongoDB(payout.account?._id?.t
oString() || payout.account.toString(), session);
      return payout;
    }));

    await Promise.all(feePayments.map(async fee => {
      await fee.save({ session });
      return fee;
    }));

    const referralPayments = [];
    if (account.referrerCode) {

referralPayments.push(...createReferralPayments(this.paymentMode
l, feePayments, payoutPayments, account, commissions[0]));
    }
    await Promise.all(referralPayments.map(async referralFee =>
{
      await referralFee.save({ session });
      return referralFee;
    }));

    if (dto.amount < account.freeCards) {
```

```
      account.freeCards = account.freeCards - dto.amount;
    } else {
      account.freeCards = 0;
    }
    await account.save({ session });

    await
this.databaseService.reCalcBalanceMongoDB(account._id.toString()
, session);

    await this.mailingService.sendTelegramNotification(
      "-852078196",
      `ZeroCard New Cards Linked: \n\nAmount:
${dto.amount}\nType: ${cardType.name}\nAccountId:
${account._id}\nTelegram: ${member.user.telegram}\nEmail:
${member.user.email}\n\nAvailable cards: \n${availableCards}`
    );

    return this.cardModel.find({ _id: { $in: cards.map(card =>
card._id) } }, {}, { session })
      .select("-__v -id -consentUrl");
  });
}
```

Код – ініціалізація депозиту.

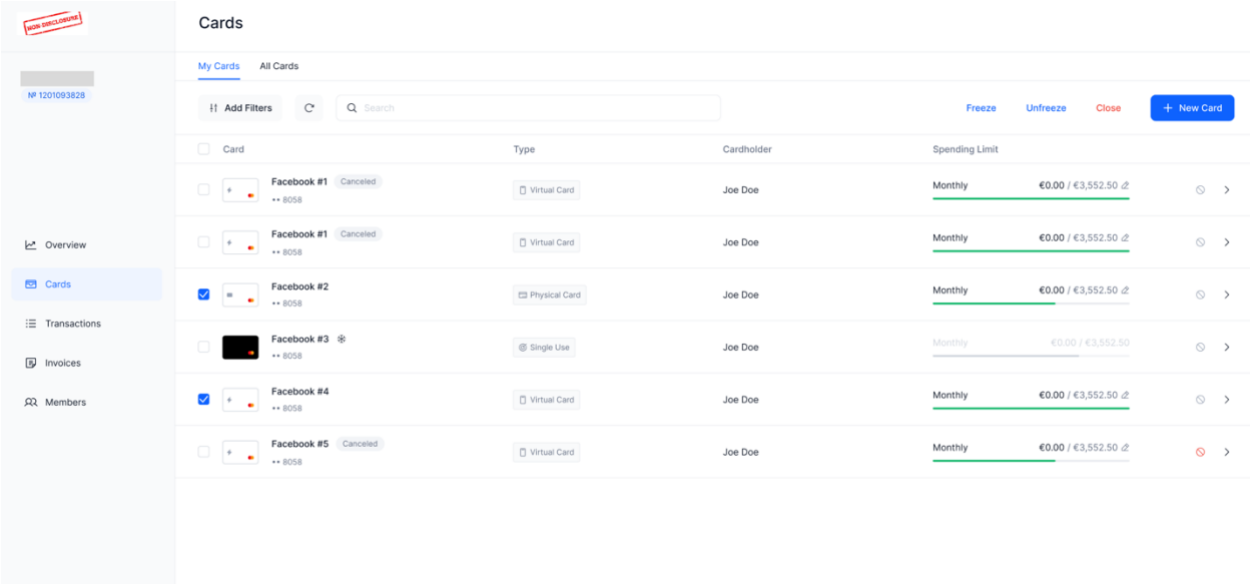
```
async initDeposit(userId: string, dto: InitDepositDto) {
  const foundByHash = await
this.paymentModel.findOne({ "cryptoData.hash": dto.hash });
  if (foundByHash) throw new HttpException("Hash already
registered", HttpStatus.BAD_REQUEST);

  return transaction(this.connection, async session => {
    const account = await
this.accountModel.findById(dto.account, {}, { session })
      .populate("referrerCode")
      .populate("legalRepresentative")
      .select("+balance +commissions")
      .populate("commissions");

    const member = await this.memberModel.findOne({ account:
account._id, user: userId }).populate({
      path: "user",
      select: "+email +telegram"
    });

    const rate = await getExchangeRate(true, this.queryService);
    const cryptoData: Crypto = {
      wallet: dto.wallet,
      network: dto.network,
      exchange_rate: rate,
      usdt_amount: Number(dto.amount.toFixed(2)),
      hash: dto.hash
```

```
    }

    const foundPaymentByHash = await
this.paymentModel.findOne({ "crypto.hash": dto.hash });
    if (foundPaymentByHash) throw new HttpException("Hash
already used", HttpStatus.BAD_REQUEST);

    const newPayment = new this.paymentModel({
      account,
      type: PaymentTypeEnum.Crypto,
      cryptoData,
      euro_amount: (dto.amount * rate).toFixed(2),
      side: TransactionSideEnum.Credit
    });

    const companies = (await
this.getAvailableCompanies(session)).map(company =>
`${company.name}: ${company.balance}`).join("\n");

    let action = "ZeroCard Deposit Request";
    if (dto.amount <= 50 ) {
      try {
        const walletData = await this.checkWallet(dto.wallet);

        if (dto.hash !== walletData.hash) throw new
HttpException("Not valid hash", HttpStatus.BAD_REQUEST);

        newPayment.euro_amount = Number((walletData.amount *
rate).toFixed(2));
        newPayment.status = PaymentStatusEnum.Approved;

        action = "ZeroCard Deposit AUTO-APPROVED";


        const commissionType = CommissionTypeEnum.Deposit;
        const commission = account.commissions.filter(commission
=> commission.type === commissionType)[0];

        const payoutPayments: any = [], feePayments: any = [];
        const totalCommissionSum =
generatePayoutsByCommission(account, this.paymentModel,
commission, 1, payoutPayments, feePayments,
newPayment.euro_amount);

        await Promise.all(payoutPayments.map(async payout => {
          await payout.save({ session });
          await
this.databaseService.reCalcBalanceMongoDB(payout.account?._id?.t
oString() || payout.account.toString(), session);
          return payout;
        }));

        if (totalCommissionSum) {
```

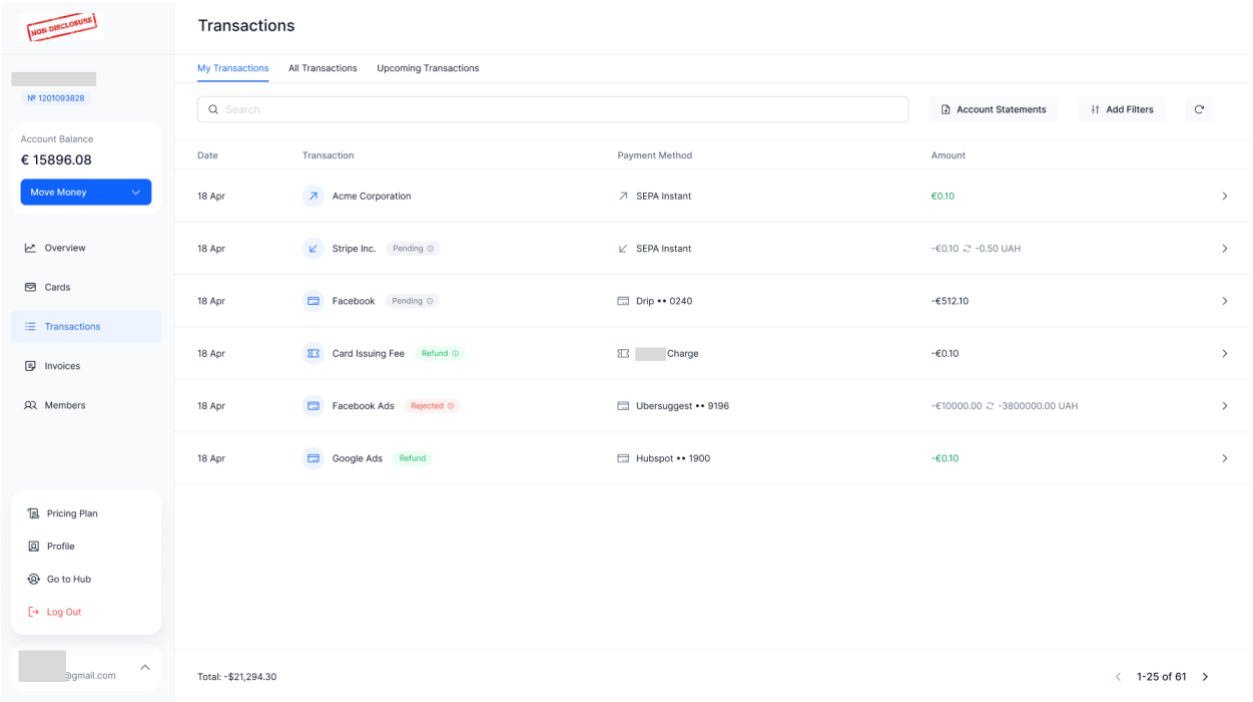
```
          await Promise.all(feePayments.map(async fee => {
            await fee.save({ session });
            return fee;
          }));

          const referralPayments = [];
          if (account.referrerCode) {

referralPayments.push(...createReferralPayments(this.paymentMode
l, feePayments, payoutPayments, account, commission));

            await Promise.all(referralPayments.map(async
referralFee => {
              await referralFee.save({ session });
              return referralFee;
            }));
            const referrerPayment = referralPayments.filter(paym
=> paym.side === TransactionSideEnum.Credit)[0];
            await
this.databaseService.reCalcBalanceMongoDB(referrerPayment.accoun
t.toString(), session);
          }
        }
      } catch (err) { console.log(err) }
    }

    await newPayment.save({ session });
    await
this.databaseService.reCalcBalanceMongoDB(account?._id?.toString
(), session);

    await this.mailingService.sendTelegramNotification(
      "-852078196",
      `${action}: \n\nAmount in USDT:
${cryptoData.usdt_amount}\nAmount in EURO:
${newPayment.euro_amount}\nWallet:
${cryptoData.wallet}\nAccountId: ${dto.account}\nTelegram:
${member?.user?.telegram || "null"}\nEmail:
${member?.user?.email || "null"}\nCompanies:\n${companies}`
    );

    return { payment: newPayment }
  })
}
```

Додаток Б – UI-інтерфейс програм

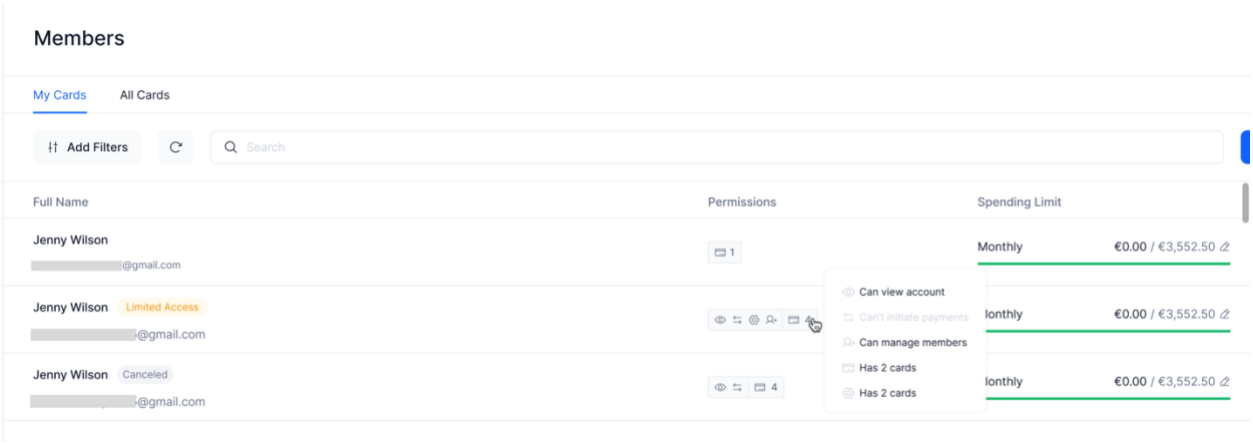
Інтерфейс сторінки управління віртуальними картками.



Інтерфейс сторінки перегляду транзакцій.



Інтерфейс сторінки управління командою рахунку.


Members
My Cards
All Cards
Add Filters
Search
Full Name
Permissions
Spending Limit
Jenny Wilson
@gmail.com
1
Monthly
€0.00 / €3,552.50
Jenny Wilson
Limited Access
@gmail.com
Can view account
Can't initiate payments
Can manage members
Has 2 cards
Has 2 cards
lonthly
€0.00 / €3,552.50
Jenny Wilson
Canceled
@gmail.com
4
lonthly
€0.00 / €3,552.50