НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет/(ННІ)

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету (Директор ННІ)	**Завідувач кафедри**
інформаційних технологій	комп'ютерних наук
(назва факультету (ННІ))	(назва кафедри)
_____ Ігор БОЛБОТ	_____ Белла ГОЛУБ
(підпис) (ім'я ПРІЗВИЩЕ)	(підпис) (ім'я ПРІЗВИЩЕ)
"___" _____ 2025 р.	"___" _____ 2025 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Аналіз та впровадження криптографічних систем захисту фінансових операцій

Спеціальність
121 Інженерія програмного забезпечення
(код і найменування)

Освітня програма
Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми
к.ф.-м.н., доцент _____ Віктор КИРИЧЕНКО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи
к.т.н., доцент _____ Ганна ВАЙГАНГ
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Виконала _____ Аліна КОРСАКОВА
(підпис) (ім'я ПРІЗВИЩЕ здобувача)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ)
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
к.т.н., доцент _____ Белла ГОЛУБ
(науковий ступінь, вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)
"____" _____20_____року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Корсаковій Аліні Вадимівні
(прізвище, ім'я, по батькові)

Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)

Освітня програма Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи Аналіз та впровадження криптографічних систем захисту фінансових операцій
затверджена наказом від "01" листопада 2024р. № 1963 «С»

Термін подання завершеної роботи на кафедру 28.11.2025
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи міжнародні стандарти інформаційної безпеки, криптографічні протоколи, що застосовуються у фінансових системах, прикладні кейси впровадження засобів криптографічного захисту в банківських і платіжних системах України та світу, які базуються на відкритих даних і аналітичних звітах щодо сучасних кіберзагроз у сфері електроних фінансових операцій.

Перелік питань, що підлягають дослідженню:
1. Основні кіберзагрози у фінансових операціях
2. Аналіз методів та систем криптографічного захисту фінансових операцій
3. Моделювання системи криптографічного захисту
4. Розробка криптографічної системи захисту фінансових операцій
5. Результати дослідження

Перелік графічного матеріалу (за потреби) презентація, постер

Дата видачі завдання "___"_____20____ р.

Керівник магістерської кваліфікаційної роботи _____ Ганна ВАЙГАНГ
(підпис) (ім'я ПРІЗВИЩЕ)

Завдання прийняла до виконання _____ Аліна КОРСАКОВА
(підпис) (ім'я ПРІЗВИЩЕ)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AES – Advanced Encryption Standard, стандарт симетричного алгоритму шифрування

API – Application Programming Interface, програмний інтерфейс застосунків

CA – Certificate Authority, центр сертифікації

CBC – Cipher Block Chaining, режим шифрування з ланцюгуванням блоків

CFB – Cipher Feedback Mode, режим зворотного зв’язку за шифротекстом

CTR – Counter Mode, режим шифрування з лічильником

DH – Diffie–Hellman, криптографічний протокол обміну ключами

DNSSEC – Domain Name System Security Extensions, розширення безпеки DNS

DSA – Digital Signature Algorithm, алгоритм цифрового підпису

ECC – Elliptic Curve Cryptography, асиметричний криптографічний метод шифрування на еліптичних кривих

ECDH – Elliptic Curve Diffie–Hellman, протокол обміну ключами на еліптичних кривих

ECDSA – Elliptic Curve Digital Signature Algorithm, цифровий підпис на еліптичних кривих

ELG – ElGamal, асиметричний криптографічний алгоритм

FHE – Fully Homomorphic Encryption, повністю гомоморфне шифрування

GCM – Galois/Counter Mode, режим шифрування з автентифікацією

HMAC – Hash-based Message Authentication Code, код автентифікації повідомлення

HTTPS – HyperText Transfer Protocol Secure, протокол захищеного передавання гіпертексту

ISO – International Organization for Standardization, Міжнародна організація стандартизації

MITM – Man-in-the-Middle, атака «людина посередині»

NIST – National Institute of Standards and Technology, Національний інститут стандартів і технологій

OTP – One-Time Password, одноразовий пароль

PBKDF2 – Password-Based Key Derivation Function 2, функція отримання ключів з пароля

PGP – Pretty Good Privacy, система захищеного обміну даними

PKI – Public Key Infrastructure, інфраструктура відкритих ключів

RSA – Rivest–Shamir–Adleman, асиметричний алгоритм шифрування

SHA – Secure Hash Algorithm, сімейство криптографічних хеш-функцій

SMIME / S/MIME – Secure/Multipurpose Internet Mail Extensions, розширення захищеної електронної пошти

SMTP – Simple Mail Transfer Protocol, протокол передавання електронної пошти

SSL – Secure Sockets Layer, протокол захищеного передавання даних

TLS – Transport Layer Security, протокол безпечної передачі даних

URL – Uniform Resource Locator, уніфікований локатор ресурсу

VPN – Virtual Private Network, віртуальна приватна мережа

WAF – Web Application Firewall, брандмауер веб-застосунків

XSS – Cross-Site Scripting, міжсайтовий скриптинг

ВСТУП

Сучасний фінансовий сектор зазнає інтенсивної цифрової трансформації, що супроводжується зростанням кількості електронних транзакцій та ускладненням потоків фінансових даних. Разом із підвищенням рівня автоматизації суттєво зростають ризики несанкціонованого доступу, підроблення інформації та порушення конфіденційності, що обумовлює необхідність застосування надійних механізмів криптографічного захисту [7; 8; 14]. Класичні підходи до безпеки стають недостатніми в умовах масштабованих онлайн-сервісів, розвитку фінтех-інфраструктур та появи квантових обчислень, здатних спричинити вразливості у традиційних криптографічних системах [1; 24]. Тому дослідження сучасних криптографічних методів і протоколів, що забезпечують стійкість фінансових транзакцій до новітніх загроз, є надзвичайно актуальним.

Криптографічні механізми становлять основу більшості протоколів безпечного обміну даними, таких як SSL/TLS, HTTPS, PGP, а також цифрових підписів, блокчейн-інфраструктур і смарт-контрактів [6; 25]. Їх ефективність визначає рівень довіри до електронних платіжних систем, сервісів онлайн-банкінгу та платформ електронної комерції. З огляду на швидкі темпи розвитку криптоаналітики, підвищення продуктивності атак і появу нових типів загроз критично важливо забезпечити адаптивність, стійкість та відповідність криптографічних рішень сучасним стандартам інформаційної безпеки.

Об’єктом дослідження є інформаційні процеси та системи, що забезпечують виконання та захист електронних фінансових транзакцій.

Предметом дослідження є криптографічні методи, алгоритми та протоколи, спрямовані на забезпечення конфіденційності, цілісності, автентичності та незаперечності фінансових даних.

Метою дослідження є аналіз сучасних криптографічних механізмів захисту фінансових транзакцій та визначення ефективних підходів до їх впровадження у фінансових ІТ-системах.

Для реалізації поставленої мети сформульовано такі **завдання**:

– проаналізувати актуальні кіберзагрози та вектори атак, що становлять ризики для електронних фінансових операцій;

– дослідити принципи роботи симетричних, асиметричних та гібридних криптографічних алгоритмів;

– здійснити оцінювання криптографічних протоколів SSL/TLS, HTTPS, PGP, а також механізмів захисту в блокчейн-системах;

– узагальнити практичний досвід застосування криптографічних технологій у банківській сфері, платіжних сервісах та електронній комерції;

– розробити рекомендації щодо інтеграції криптографічних рішень у фінансову інфраструктуру з урахуванням вимог безпеки та продуктивності.

Для досягнення мети дослідження застосовано **методи системного та порівняльного аналізу**, криптографічного моделювання, ризик-орієнтованого оцінювання, аналізу стійкості алгоритмів та протоколів до відомих типів атак, а також методи експертного оцінювання ефективності криптографічних рішень у прикладних фінансових системах.

Наукова новизна роботи полягає у комплексному аналізі застосування сучасних криптографічних механізмів у фінансових транзакціях, у визначенні критеріїв вибору ефективних алгоритмів та протоколів для різних типів фінансових операцій, а також у формуванні рекомендацій щодо інтеграції криптографічних технологій у фінансову інфраструктуру з урахуванням актуальних загроз та вимог ринку.

Апробація результатів здійснювалась у процесі обговорень із науковим керівником, під час підготовки матеріалів до участі у наукових конференціях з інформаційної безпеки, а також у рамках навчальних досліджень, де перевірялась практична застосовність запропонованих рішень.

Практичне значення дослідження полягає в можливості використання його результатів для підвищення інформаційної безпеки у фінансових інформаційних системах шляхом впровадження сучасних криптографічних

рішень у банківські ІТ-системи, платіжні платформи та політики захисту даних [11; 20; 25]. Отримані напрацювання також можуть бути корисними в освітньому та дослідницькому процесі для підготовки фахівців у сфері кібербезпеки та створення відповідного навчального забезпечення.

Структура магістерської роботи включає вступ, три розділи, висновки, список використаних джерел та додатки. Перший розділ містить теоретичний огляд криптографічних методів та аналіз загроз. У другому розділі досліджено алгоритми шифрування та криптографічні протоколи, оцінено їх стійкість і сфери застосування. Третій розділ присвячений аналізу практичних кейсів та формуванню рекомендацій щодо застосування криптографічних рішень у фінансових системах.

РОЗДІЛ 1. ОСНОВНІ КІБЕРЗАГРОЗИ У ФІНАНСОВИХ ОПЕРАЦІЯХ

1.1 Актуальність проблеми кібербезпеки у фінансовій сфері

Фінансова сфера сьогодні фактично тримає на собі основу цифрової економіки – через неї постійно рухається капітал, укладаються міжнародні угоди, формується стабільність цілих держав. Проте саме через цю важливість галузь часто опиняється в центрі уваги кіберзлодіїв. У банках та платіжних системах зберігаються гігантські обсяги чутливих даних: і про клієнтів, і про компанії.

Перехід на електронні транзакції, онлайн-банкінг та мобільні додатки зробив користування грошима простішим, але й відкрив купу нових «дір». За звітом Європейського агентства з кібербезпеки (ENISA), фінансові установи досі у топі цілей для атак [8]. У межах одного лише періоду в ЄС зафіксували понад 480 випадків, і майже половина – це саме банки. У списку типових загроз – фішинг, DDoS, шкідливе ПЗ (Emotet, Qbot) і крадіжка паролів [7; 8].

Компанія IBM у своєму звіті теж підтверджує: злочинці часто не витрачають зусилля на складний злам, а просто отримують доступ до чинних акаунтів. Приблизно 30 % усіх інцидентів саме такого типу [14]. Це свідчить про зміну підходів – менше «грубого» злому, більше соціальної інженерії, фішингу й маніпуляцій із довірою користувачів.

Методи атак, природно, не стоять на місці. Якщо колись найбільше боялися банківських троянів, то нині на перший план виходять складніші прийоми: *Man-in-the-Middle*, підробка платіжних запитів, компрометація API або автоматизований фішинг за допомогою ШІ [8]. Чимало таких атак цілеспрямовані – їх планують під конкретну компанію чи навіть людину.

Інколи збитки не обмежуються грошима. Падає довіра, йдуть клієнти, репутація сиплеться, а далі – і фінансова нестабільність. Тому інформаційна безпека перестала бути суто технічною справою.

Звідси логічно випливає потреба у криптографічному захисті. Алгоритми AES-256 чи ECC, протоколи TLS 1.3, HTTPS, VPN, PGP – це вже не просто технічні терміни, а основа безпечного обміну даними [4; 13; 6]. Не менш важливими залишаються правильне управління ключами, двофакторна автентифікація й апаратні модулі безпеки (HSM) [18; 20].

Тож проблема кіберзахисту у фінансовій сфері сьогодні стоїть гостро як ніколи. Реальне рішення можливе лише за умови поєднання технологій, правил і навчання користувачів. Криптографія при цьому – не просто інструмент, а фундамент довіри в цифровій економіці.

1.2 Класифікація кіберзагроз для фінансових операцій

Щоб система кіберзахисту працювала реально, а не лише формально, спершу потрібно розібратися, від чого саме вона має захищати. Загрози різні – і за походженням, і за тим, як саме вони проявляються. У таблиці 1.1 наведено узагальнену схему класифікації ризиків, притаманних фінансовим операціям. Вона охоплює як зовнішні чинники, так і внутрішні слабкі місця, які можуть вплинути на цілісність або конфіденційність даних.

Таблиця 1.1

Класифікація кіберзагроз

Група загроз	Тип загрози	Короткий опис
Соціальна інженерія	Фішинг / цільовий фішинг	Підроблені листи або повідомлення, що імітують банки чи платіжні сервіси з метою викрадення логінів, паролів, OTP-кодів.
	Телефонне та SMS-шахрайство (vishing, smishing)	Зловмисники телефонують або надсилають SMS, видаючи себе за працівників банку, щоб отримати дані клієнта чи авторизаційні коди.
	Pretexting (створення легенди)	Маніпуляції довірою, коли злочинець представляється партнером, аудитом чи службою безпеки для отримання інформації.

Група загроз	Тип загрози	Короткий опис
Шкідливе ПЗ та технічні атаки	Банківські трояни (*Dridex*, *TrickBot*)	Програми, що викрадають облікові дані, підміняють реквізити транзакцій або контролюють сесії інтернет-банкінгу.
	Програми-вимагачі (ransomware)	Шифрують бази даних і сервери банку, блокуючи доступ до транзакцій чи клієнтських даних до сплати викупу.
	Man-in-the-Browser / MITM	Перехоплення або зміна даних під час фінансових транзакцій у браузері чи мережевому з'єднанні.
	Експлуатація вразливостей (zero-day, unpatched systems)	Використання технічних недоліків ПЗ банків чи платіжних шлюзів для отримання несанкціонованого доступу.
Атаки на інфраструктуру та сервіси	DDoS-атаки	Перевантаження серверів банків чи платіжних систем, що призводить до зупинки онлайн-банкінгу або затримки операцій.
	Атаки на хмарні сервіси та API	Використання помилок конфігурації або викрадених ключів для доступу до фінансових даних у хмарі.
	Supply-chain атаки	Компрометація ПЗ чи сервісів постачальників, що інтегровані у фінансову інфраструктуру.
	Атаки на платіжні системи (SWIFT, POS, банкомати)	Підміна або інжекція платіжних запитів, зараження банкоматів, скімінг даних карт.
Компрометація облікових даних і автентифікації	Account takeover (захоплення акаунтів)	Отримання контролю над обліковими записами клієнтів або працівників для ініціювання шахрайських операцій.
	Викрадення або підробка криптографічних ключів	Доступ до зашифрованих транзакцій або підробка цифрових підписів для підтвердження платежів.

Кожен тип загроз має власну специфіку. Технічні – зазвичай це помилки в інфраструктурі, уразливості програм чи мережевих сервісів. Соціальна інженерія, навпаки, працює з людьми: вона використовує не код, а довіру, не

системи, а психологію користувача. Є й комбіновані атаки – справжні «гібриди», що поєднують кілька методів одночасно. Такі дії часто залишаються непоміченими, іноді навіть місяцями.

Отже, класифікація кіберзагроз – це не просто перелік термінів, а база, без якої неможливо спланувати адекватну оборону. Вона допомагає визначити пріоритети, розподілити ресурси та побудувати стратегію, у якій технічні, організаційні й процедурні заходи доповнюють одне одного.

1.3 Типові атаки на банківські системи та платіжні сервіси

У попередньому підрозділі було запропоновано класифікацію кіберзагроз, притаманних фінансовим операціям. Проте сама по собі така класифікація не має практичної користі, якщо не розглядати її у контексті реальних ситуацій. Адже саме під час щоденної роботи банків, фінтех-компаній і платіжних сервісів проявляється, як теоретичні загрози перетворюються на конкретні інциденти.

1. Соціальна інженерія: атаки на клієнтів і персонал

Соціальна інженерія залишається одним із найстаріших і найдієвіших способів зламу. Її сила не у технології, а у впливі на людину. Зловмисник переконує жертву самостійно відкрити двері системі: надати пароль, підтвердити переказ або натиснути на «безпечне» посилання.

Найпоширенішим є фішинг – розсилання підроблених листів або SMS, які виглядають як офіційні повідомлення банку [14]. Людина бачить знайомий логотип, переходить за посиланням і вводить свої дані – логін, пароль чи одноразовий код. Далі все просто: інформація опиняється у зловмисників.

Більш витончений варіант – *spear-phishing*: персоналізована атака на конкретного співробітника. У листі можуть бути справжні дані про підрозділ чи посаду, тож повідомлення виглядає правдоподібно.

До цього додаються й інші схеми – *vishing* (телефонні дзвінки від “служби безпеки банку”) та *smishing* (SMS про нібито підозрілу операцію). Такі методи б’ють по найслабшому – страху втратити гроші.

2. Шкідливе програмне забезпечення

Malware залишається основним інструментом кібершахраїв. Потрапити до системи воно може різними шляхами – від зараженого вкладення до компрометованого оновлення легітимного програмного продукту.

Банківські трояни – Dridex, TrickBot, Qbot, Emotet – інтегруються у браузери, фіксують натискання клавіш, перехоплюють cookies і навіть підмінюють реквізити під час платежів [7; 8].

Іноді використовується техніка *Man-in-the-Browser*, коли користувач бачить правильні дані, а банк отримує змінені. Без цифрових підписів або криптографічної перевірки виявити це майже неможливо.

Окремий вид – ransomware, або програми-вимагачі. Вони шифрують дані та вимагають викуп у криптовалюті. Часто застосовується «подвійний шанта»: крім шифрування, дані ще й копіюють, погрожуючи їх оприлюднити.

Крім того, у банківських системах поширені *keyloggers*, *spyware* та *backdoors*, які дозволяють шпигувати за діями користувачів або змінювати налаштування без помітних слідів.

3. Атаки на інфраструктуру й процесингові сервіси

Банківська інфраструктура – це складна мережа серверів, каналів зв’язку, баз даних, API-інтерфейсів та хмарних сервісів. Вона забезпечує безперервність платежів і є очевидною мішенню для нападників.

Найбільш відомі – DDoS-атаки. Потоки запитів перевантажують сервери, і сервіси стають недоступними. Часто це лише «димова завіса» для іншої, складнішої операції.

Сьогодні, із розвитком *open banking* і хмарних технологій, різко зросла кількість атак через помилки конфігурації або публікацію ключів API. Також поширені *supply-chain* атаки, коли ураження відбувається через скомпрометоване оновлення легального ПЗ .

Класичний приклад – злам системи SWIFT у Бангладеш 2016 року, коли зловмисники ініціювали фальшиві перекази на 81 млн доларів [3]. Подібні випадки зафіксовані й у В’єтнамі, Еквадорі та на Філіппінах.

4. Компрометація облікових записів і ключів

Захоплення облікових записів (*Account Takeover*) – одна з найсерйозніших проблем. Після фішингу або витоку баз даних у руках зловмисників опиняються логіни, паролі, токени доступу [14]. Це дає змогу діяти від імені користувача, змінювати налаштування рахунків і навіть створювати нові шаблони платежів.

Ще небезпечніше – втрата контролю над криптографічними ключами. Якщо приватний ключ опиняється у сторонніх осіб, вони можуть видавати фальшиві сертифікати, підробляти цифрові підписи чи розшифровувати захищений трафік.

Причини банальні: застарілі алгоритми, неправильне зберігання ключів, тощо. У великих організаціях ключі іноді передаються між підрозділами навіть у незашифрованому вигляді, що створює серйозні ризики.

Отже, типові кібератаки на банківські системи та платіжні сервіси – це не просто теоретичні приклади з підручників. Вони наочно показують, як класифіковані в попередньому розділі загрози проявляються у реальному фінансовому середовищі. Сучасний банківський сектор фактично живе в умовах постійної боротьби: кіберзлочинці змінюють тактику, комбінують технічні прийоми з психологічним тиском і шукають найслабші місця в захисті.

Найпідступнішими вважаються змішані, багаторівневі атаки. Зазвичай усе починається з соціальної інженерії – листа, дзвінка чи повідомлення, яке викликає довіру. Потім у хід іде шкідливе програмне забезпечення, що закріплюється у системі та розширює доступ. І вже на фінальному етапі зловмисники використовують вразливості інфраструктури, щоб отримати гроші або викрасти конфіденційні дані [3; 10]. На практиці такі атаки нагадують ланцюгову реакцію, коли одна помилка користувача запускає цілу серію подій.

1.4 Методи нейтралізації та попередження кіберзагроз

У сучасному фінансовому середовищі питання кіберзахисту вже не зводиться лише до технологій. Ефективна оборона базується на поєднанні

технічних, організаційних, правових і навіть освітніх заходів [15; 16]. Без цього будь-яка, навіть найдосконаліша система, залишиться вразливою.

Головна мета кіберзахисту – не просто ліквідувати наслідки інцидентів, а попередити їх. На практиці це означає створення такої архітектури безпеки, у якій напад стає надто складним або невигідним для зловмисника. Саме на цьому принципі побудована концепція багаторівневого захисту (*defense in depth*). Її суть – у взаємодії кількох бар'єрів, що перекривають один одного.

Технічна складова включає шифрування даних, ідентифікацію та автентифікацію користувачів, фільтрацію трафіку, контроль доступу, резервне копіювання і системи виявлення вторгнень. Якщо одна лінія оборони не спрацьовує, інша бере удар на себе. У результаті вдається зберегти конфіденційність та цілісність даних, навіть коли атака все ж відбулася.

Організаційні заходи мають не менше значення. Це розробка політик безпеки, аудит користувачів, контроль прав адміністраторів, захист внутрішніх мереж. Такі дії, хоч і здаються «паперовими», часто запобігають найсерйознішим порушенням. Варто пам'ятати, що більшість інцидентів трапляється через нехтування правилами, а не через слабкість технологій.

Окрему увагу слід приділити людському фактору. Саме він найчастіше стає першим кроком до інциденту. Навчання персоналу, імітації фішингових атак, короткі тестування після тренінгів – це не формальність, а реальний спосіб зменшити ризики. У деяких банках подібні навчання вже стали частиною корпоративної культури, і це дало відчутні результати.

Важливо також мати системи моніторингу та аналітики. Сучасні рішення SIEM, UEBA або поведінкові фільтри дозволяють бачити підозрілі події майже в реальному часі. Іноді навіть невелика аномалія – надлишковий запит, підозрілий логін чи нетипова поведінка користувача – може вказати на атаку.

Профілактика охоплює керування вразливостями: оновлення програмного забезпечення, аудит, пентести. Це буденна, але необхідна робота, без якої неможливо підтримувати систему в актуальному стані.

Нарешті, особливе місце посідає криптографічний захист. Він забезпечує цілісність, автентичність і конфіденційність фінансових даних під час обміну. Навіть якщо інформацію перехоплено, без ключів вона залишається непридатною для читання.

Таким чином, справжня кібербезпека у фінансових системах – це поєднання технологій, правил і навчання. І хоча жодна система не гарантує абсолютного захисту, багаторівневий підхід значно ускладнює життя зловмисникам і підвищує довіру користувачів до цифрових фінансових сервісів.

РОЗДІЛ 2. АНАЛІЗ МЕТОДІВ ТА СИСТЕМ КРИПТОГРАФІЧНОГО ЗАХИСТУ ФІНАНСОВИХ ОПЕРАЦІЙ

2.1 Роль, мета та завдання криптографії у забезпеченні безпеки фінансових операцій

Криптографія сьогодні є одним із ключових засобів захисту інформації у фінансовій сфері. Без неї сучасні банки, платіжні системи та фінтех-платформи не змогли б гарантувати безпечну роботу з даними. Вона формує основу цифрової довіри, адже будь-яка транзакція або запит користувача має бути захищена від перехоплення й підміни.

У практичному використанні криптографія виконує кілька важливих завдань. По-перше, вона забезпечує конфіденційність, тобто приховує зміст даних від сторонніх осіб. По-друге, гарантує цілісність, коли будь-яке втручання в інформацію можна виявити. І нарешті, відповідає за автентифікацію та невідмовність, підтверджуючи, що дія виконана саме тим користувачем, який її ініціював.

Одним із найпоширеніших напрямів є захист каналів передавання даних. Протоколи TLS, HTTPS, SSL та SSH створюють безпечне середовище обміну між клієнтом і сервером, шифруючи інформацію ще під час відправлення [6; 25]. Це дозволяє безпечно здійснювати онлайн-платежі, користуватися мобільним банкінгом і працювати з конфіденційними документами навіть у публічних мережах.

Криптографічний захист фінансових даних – це не просто набір технічних засобів, а система алгоритмів, що гарантують конфіденційність, автентичність і цілісність інформації. Математичні перетворення, на яких він базується, роблять дані непридатними для читання без ключа розшифрування. Тож навіть якщо

хтось перехопить повідомлення, без ключа воно залишиться звичайною послідовністю символів.

Головна мета криптографічного захисту – створення середовища, у якому фінансова інформація може передаватися та зберігатися без ризику втрати чи розголошення. Це дозволяє зменшити ризики шахрайства й підтримувати довіру між банками, сервісами й користувачами.

Основні завдання криптографії у фінансовій сфері охоплюють:

- захист конфіденційності шляхом шифрування;
- перевірку автентичності учасників операцій;
- контроль цілісності даних під час обміну;
- невідмовність – підтвердження факту виконання транзакції;
- управління ключами: генерація, зберігання, оновлення та розповсюдження.

Банки активно застосовують ці принципи у своїх системах. Клієнтські дані, історія транзакцій і резервні копії зберігаються у зашифрованому вигляді, найчастіше за допомогою симетричних алгоритмів AES або гібридних схем AES + ECC, що дозволяють поєднати швидкодію з надійністю.

Важливим компонентом є цифровий підпис, який надає документам юридичної сили. Його функціонування базується на інфраструктурі відкритих ключів (PKI) – сертифікатах, токенах і механізмах перевірки достовірності [9; 27].

Особливу роль відіграють апаратні криптографічні засоби: токени, смарт-карти, HSM-модулі. Вони зберігають приватні ключі локально у користувача та виконують операції підпису без передачі секретної інформації у відкритому вигляді. У поєднанні з багатофакторною автентифікацією це формує надійний багаторівневий захист.

Сьогодні криптографія виходить за межі класичних банківських систем і активно використовується у блокчейні, цифрових валютах, смарт-контрактах. У цих технологіях вона не лише захищає дані, а й забезпечує прозорість і незмінність записів. Наприклад, у блокчейні кожна транзакція має власний

криптографічний підпис, а структура ланцюга виключає можливість підробки інформації.

Отже, криптографія – це основа цифрової безпеки й довіри. Вона супроводжує фінансові дані на всіх етапах їхнього життєвого циклу, забезпечуючи стабільність і захист інформаційних процесів у сучасній економіці.

2.2 Класифікація криптографічних систем

Криптографічні системи становлять основу будь-якої інфраструктури захисту інформації у фінансовій сфері. Вони використовуються для забезпечення безпечного обміну даними, автентифікації користувачів, підтвердження транзакцій, створення цифрових підписів і запобігання підробці фінансових документів. Сучасна фінансова індустрія – від банківських процесингових центрів до систем мобільного банкінгу – спирається на криптографічні механізми, що забезпечують цілісність, конфіденційність і довіру у взаємодії між усіма учасниками [25].

1. Класифікація за типом ключів

Найпоширенішою основою поділу криптографічних систем є тип використовуваних ключів, який визначає спосіб шифрування і розшифрування інформації. Залежно від цього параметра виділяють симетричні, асиметричні та гібридні системи [17].

Симетричні криптосистеми використовують один і той самий секретний ключ для шифрування та дешифрування даних. Це забезпечує високу швидкодію і робить такі алгоритми зручними для обробки великих обсягів фінансової інформації – зокрема, даних про платежі, звітів або архівів транзакцій. Симетричні алгоритми стали стандартом у банківських платформах і мережевих шлюзах. Їх головним недоліком є проблема безпечного розповсюдження ключів, адже якщо ключ потрапляє до зловмисника, він може розшифрувати всі передані повідомлення.

Асиметричні криптосистеми, навпаки, ґрунтуються на використанні двох математично пов’язаних ключів – відкритого (public) і закритого (private). Відкритий ключ може вільно передаватися іншим сторонам, тоді як закритий зберігається в таємниці у власника. Дані, зашифровані відкритим ключем, можуть бути розшифровані лише відповідним закритим, що робить такий підхід ідеальним для відкритих мереж і онлайн-транзакцій. Вони широко застосовуються у платіжних протоколах (TLS, HTTPS), системах електронного підпису, а також у токенах для автентифікації клієнтів [6].

На практиці банки та фінансові установи використовують гібридні криптосистеми, що поєднують обидва підходи: асиметричні алгоритми застосовуються для безпечного обміну ключами, а симетричні – для шифрування великих масивів даних. Така модель поєднує високу швидкість із надійністю та зменшує ризики компрометації.

В основі побудови криптографічних рішень для фінансової сфери лежить баланс між продуктивністю та криптостійкістю. Симетричні алгоритми забезпечують високу швидкість шифрування, але мають складність із безпечним розповсюдженням ключів; асиметричні – навпаки, гарантують безпечний обмін ключами, проте мають високу обчислювальну вартість; гібридні – поєднують переваги обох класів. У таблиці 2.1 узагальнено проведений порівняльний аналіз криптографічних систем різних типів.

Таблиця 2.1

Порівняння криптографічних систем різних типів

Критерій	Симетричні (AES, Twofish, ChaCha20)	Асиметричні (RSA, ECC)	Гібридні (TLS, PGP)
Швидкодія	Найвища, шифрування великих масивів даних у реальному часі	Дуже низька (RSA), середня (ECC)	Залежить від симетричної частини
Стійкість до атак	Висока за умови довжини ключа ≥ 128 біт	Висока, ECC має кращу стійкість при коротшому ключі	Висока завдяки комбінуванню

Критерій	Симетричні (AES, Twofish, ChaCha20)	Асиметричні (RSA, ECC)	Гібридні (TLS, PGP)
Вимоги до пам’яті	Низькі, придатні для мобільних пристроїв	Підвищені (RSA), середні (ECC)	Середні
Область застосування	Шифрування каналів, зберігання даних	Підпис, обмін ключами, автентифікація	Банківські протоколи (TLS), e-commerce

Отже, представлене порівняння демонструє, що кожен тип криптографічної системи має власну сферу оптимального застосування, зумовлену співвідношенням швидкодії, рівня стійкості та ресурсних вимог. Симетричні алгоритми є найефективнішими для обробки великих обсягів транзакційних даних завдяки малій затримці та низьким вимогам до пам’яті, проте їх використання потребує надійних механізмів управління ключами. Асиметричні системи забезпечують високий рівень автентичності й інтегруються у PKI-інфраструктуру, однак їх обчислювальна складність ускладнює використання для потокового шифрування. Гібридні моделі, які поєднують сильні сторони обох підходів, демонструють найбільш збалансовані властивості в умовах фінансових операцій з підвищеними вимогами до продуктивності та безпеки. Результати порівняльного аналізу підтверджують доцільність їх застосування в сучасних платіжних протоколах і банківських інформаційних системах, де важливими є як швидкодія, так і криптостійкість, а також стійкість до атак на ключову інфраструктуру [6; 20].

2. Класифікація за призначенням

За функціональним призначенням криптографічні системи поділяються залежно від того, яку конкретну задачу інформаційної безпеки вони вирішують. У межах фінансових операцій такі системи можуть виконувати різні ролі, утворюючи багаторівневу архітектуру захисту.

За функціональним призначенням криптографічні системи поділяються на кілька основних груп:

а. Системи забезпечення конфіденційності – відповідають за збереження таємниці даних під час їх передавання або зберігання. Вони гарантують, що

інформація про платіж, клієнта чи операцію залишається недоступною для сторонніх осіб навіть у разі перехоплення трафіку чи компрометації каналу зв’язку.

b. Системи автентифікації – забезпечують підтвердження особи користувача, банку чи фінансового сервісу. Їх робота базується на використанні цифрових сертифікатів, криптографічних токенів, біометричних ідентифікаторів або пар ключів, що гарантують справжність суб’єкта транзакції.

c. Системи контролю цілісності – запобігають несанкціонованим змінам або пошкодженню даних у процесі обробки. Для цього використовуються криптографічні хеш-функції, які формують унікальний цифровий відбиток інформації та дозволяють перевірити її незмінність.

d. Системи електронного цифрового підпису (ЕЦП) – забезпечують юридичну значущість фінансових операцій, підтверджують авторство транзакції та гарантують невідмовність дій користувача. ЕЦП активно застосовується у міжбанківських розрахунках, системах дистанційного обслуговування клієнтів і документообігу.

e. Системи управління ключами (Key Management Systems, KMS) – становлять інфраструктурний компонент криптографічного захисту, який відповідає за генерацію, розповсюдження, оновлення та знищення ключів. Надійна система управління ключами підтримує цілісність усієї криптографічної екосистеми та мінімізує ризики компрометації.

3. Класифікація за способом реалізації

Ще одним важливим критерієм класифікації є спосіб технологічного впровадження криптографічної системи, який визначає, у якому середовищі виконуються обчислення та яким чином організовано зберігання криптографічних ключів. Відповідно до цього виділяють програмні, апаратні та комбіновані системи.

Програмні системи реалізуються у вигляді програмних бібліотек або модулів, що інтегруються у банківські додатки, сервери обробки платежів чи мобільні фінансові сервіси. Їх головна перевага – гнучкість і можливість швидко

оновлювати алгоритми відповідно до нових стандартів безпеки. Наприклад, такі бібліотеки, як OpenSSL чи BouncyCastle, використовуються для реалізації протоколів захисту у фінансових вебсервісах.

Апаратні системи функціонують на базі спеціалізованих пристроїв, таких як HSM (Hardware Security Module) або TPM (Trusted Platform Module). Вони фізично ізольовують ключі від зовнішнього середовища, що робить їх практично недоступними для хакерів навіть у разі компрометації операційної системи. Такі рішення широко застосовуються у банках, де проводяться критичні операції, наприклад, підпис транзакцій SWIFT чи генерація токенів для клієнтів.

Комбіновані системи поєднують обидва підходи: програмна частина відповідає за логіку транзакцій і взаємодію з клієнтами, тоді як апаратна – за генерацію та зберігання ключів [18; 20]. Це дозволяє збалансувати продуктивність, зручність і рівень безпеки, що особливо важливо у масштабних фінансових екосистемах.

4. Класифікація за структурою обробки інформації

Ще одним аспектом є структурна організація процесу шифрування, тобто спосіб, у який інформація обробляється під час захисту. За цим критерієм криптографічні системи поділяються на блокові, потокові та змішані.

У блокових системах інформація розділяється на послідовні блоки фіксованої довжини, кожен з яких шифрується окремо [4]. Такий підхід дозволяє ретельно контролювати кожен етап обробки й забезпечує високу точність захисту.

Потокові системи працюють у режимі реального часу, перетворюючи інформацію побітово або побайтово. Вони особливо ефективні у мобільних додатках і системах швидких платежів, де важливими є мінімальні затримки.

Змішані системи можуть автоматично перемикатися між блоковим і потоковим режимом залежно від типу даних і швидкісних вимог, що характерно для сучасних платіжних шлюзів і систем цифрових валют.

5. Класифікація за рівнем інтеграції

Залежно від масштабу використання виділяють локальні, мережеві та глобальні (розподілені) криптографічні системи.

Локальні системи захищають внутрішні дані організації: бази клієнтів, бухгалтерські записи, резервні копії.

Мережеві системи забезпечують безпеку під час обміну між різними сервісами – наприклад, між банківським додатком і процесинговим центром.

Розподілені системи реалізуються у міжнародних платіжних протоколах (SWIFT, SEPA) або у технологіях блокчейн, де криптографія виступає гарантом довіри між незалежними учасниками без центрального контролю.

З огляду на різноманітність завдань, які вирішуються у фінансових інформаційних системах, криптографічні методи поділяють на кілька базових груп залежно від принципів побудови, механізмів керування ключами та способів обробки даних. Така класифікація є фундаментом для подальшого аналізу алгоритмів і визначення їх придатності до використання у високонавантажених платіжних середовищах, банківських процесингових центрах та системах електронного документообігу. У таблиці 2.2 представлено порівняльну характеристику основних криптографічних систем, що застосовуються у сучасній фінансовій сфері, з урахуванням їх функціональних властивостей та практичних обмежень.

Таблиця 2.2

Класифікація криптографічних систем

Критерій класифікації	Типи криптографічних систем	Характеристика
Принцип управління ключами	Симетричні, асиметричні, гібридні	Визначає логіку обміну секретною інформацією між сторонами
Функціональне призначення	Конфіденційність, автентифікація, цілісність, підпис, управління ключами	Формує основу багаторівневого захисту у фінансових системах

Критерій класифікації	Типи криптографічних систем	Характеристика
Тип реалізації	Програмні, апаратні, комбіновані	Визначає технічну платформу криптографічного захисту
Структура обробки даних	Блокові, потокові, змішані	Визначає спосіб перетворення даних у процесі захисту
Рівень інтеграції	Локальні, мережеві, розподілені	Визначає масштаб дії та архітектуру системи

Аналіз наведеної класифікації свідчить, що кожна група криптографічних систем виконує власну роль у забезпеченні комплексного захисту фінансових операцій. Симетричні алгоритми є оптимальними для шифрування значних обсягів даних у режимі реального часу, оскільки забезпечують високу швидкодію та мінімальні витрати ресурсів, проте мають складність у безпечному розповсюдженні ключів. Асиметричні системи, навпаки, гарантують захищений обмін ключами та високий рівень автентичності, але характеризуються значними обчислювальними витратами. Гібридні криптографічні схеми поєднують сильні сторони обох підходів, що дозволяє досягти балансу між продуктивністю та стійкістю, особливо у платіжних протоколах та мережевих стандартах. Таким чином, класифікація демонструє необхідність комплексного використання різних типів криптографічних систем у фінансовій інфраструктурі для забезпечення надійного та безперервного захисту транзакцій.

2.3 Основні вимоги до безпеки фінансових транзакцій

Будь-яка транзакція, незалежно від її розміру чи призначення, має здійснюватися в умовах гарантованого захисту даних від несанкціонованого доступу, підробки або втручання третіх осіб. Ефективна система захисту повинна ґрунтуватися на комплексі вимог, які визначають не лише рівень криптографічної стійкості, а й надійність усієї інфраструктури обміну фінансовою інформацією.

Основні принципи безпеки, що формують вимоги до фінансових транзакцій, традиційно визначаються через модель CIAAN – Confidentiality (конфіденційність), Integrity (цілісність), Authenticity (автентичність), Availability (доступність) та Non-repudiation (невідмовність). Вони є базою для розроблення, впровадження й оцінювання будь-яких криптографічних систем у банківській і фінансовій сферах.

1. Конфіденційність. Передбачає, що інформація про фінансову операцію доступна лише уповноваженим сторонам. Дані, пов’язані з реквізитами рахунків, сумами платежів, персональними даними клієнтів або деталями договорів, повинні бути захищені від несанкціонованого ознайомлення. Досягнення конфіденційності забезпечується за допомогою криптографічного шифрування, багатофакторної автентифікації та розмежування прав доступу. Наприклад, у системах онлайн-банкінгу канали передачі даних захищаються протоколами TLS/SSL, а доступ користувачів контролюється через одноразові паролі (OTP) або біометричну ідентифікацію [6; 25]. Втрата конфіденційності може призвести не лише до фінансових збитків, а й до репутаційних ризиків для банківської установи.

2. Цілісність. Означає збереження їх первісного стану під час обробки, передавання чи зберігання. Будь-яке спотворення інформації – навмисне чи випадкове – може призвести до неправильного виконання фінансової операції або викривлення звітності. Забезпечення цілісності досягається за допомогою хеш-функцій (SHA-2, SHA-3), контрольних сум і механізмів перевірки цифрового підпису, які дозволяють перевірити, чи не змінювалися дані після відправлення. У фінансових системах такі механізми інтегровані в платіжні шлюзи, процесингові центри та банківські API, що дає змогу своєчасно виявляти спроби підробки або втручання в потоки транзакцій.

3. Автентичність. гарантує, що учасники транзакції дійсно є тими, за кого себе видають, а передані дані походять із перевіреного джерела. Це надзвичайно важливо для запобігання шахрайству, фішингу, компрометації облікових записів і несанкціонованого доступу. Автентифікація у фінансових системах

здійснюється через сертифікати відкритих ключів (PKI), цифрові підписи, токени доступу, а також біометричні технології (відбиток пальця, розпізнавання обличчя). Наприклад, при здійсненні міжбанківських розрахунків використовується інфраструктура відкритих ключів (Public Key Infrastructure), яка забезпечує довіру між сторонами, навіть якщо вони фізично знаходяться у різних країнах [17].

4. Доступність. означає, що фінансова система повинна бути працездатною й готовою до обробки транзакцій у будь-який момент часу. Відмова в обслуговуванні, перевантаження або збої в роботі можуть призвести до порушення виконання платежів і фінансових зобов’язань. Для підтримання доступності застосовуються системи резервування даних, кластеризація серверів, балансування навантаження та захист від DDoS-атак. У платіжних мережах використовуються розподілені інфраструктури, які дозволяють миттєво перемикати обробку запитів на резервні сервери без втрати даних або часу виконання.

5. Невідмовність. забезпечує неможливість заперечення факту участі сторони у транзакції або підписанні документа. Іншими словами, після виконання операції жоден її учасник не може заявити, що не здійснював певних дій. Цей принцип реалізується за допомогою цифрових підписів, журналів аудиту (audit logs) та сертифікатів автентичності, які фіксують кожен етап обробки фінансових даних. У банківських системах це особливо важливо для дотримання юридичної відповідальності – наприклад, під час укладання електронних договорів, видачі кредитів або проведення великих міжбанківських переказів.

2.4 Огляд сучасних симетричних та асиметричних алгоритмів шифрування

Забезпечення конфіденційності та цілісності фінансових даних у сучасних інформаційних системах неможливе без використання ефективних алгоритмів

шифрування. Вибір криптографічного методу безпосередньо впливає на рівень безпеки транзакцій, швидкість їх обробки та стійкість до атак. У практиці фінансових установ найпоширенішими є симетричні та асиметричні алгоритми шифрування, які реалізують різні підходи до захисту даних, але часто використовуються у взаємодоповнювальній формі.

У рамках даного дослідження основна увага приділялася аналізу двох найбільш поширених і практично значущих алгоритмів – AES-256 та ECC, які сьогодні є базовими елементами криптографічного захисту у фінансових системах. Вибір саме цих алгоритмів зумовлений їхнім широким застосуванням у міжнародних стандартах безпеки, перевіреною стійкістю та ефективністю при реалізації у банківських і платіжних інфраструктурах.

1. Симетричні алгоритми шифрування.

Симетричне шифрування – це метод, у якому для зашифрування та розшифрування інформації використовується один і той самий секретний ключ. Головною перевагою такого підходу є висока швидкодія, що дозволяє обробляти великі обсяги даних у реальному часі, що особливо актуально для банківських процесингових систем і платіжних платформ. Принцип дії симетричної криптосистеми наведено на рисунку 2.1.

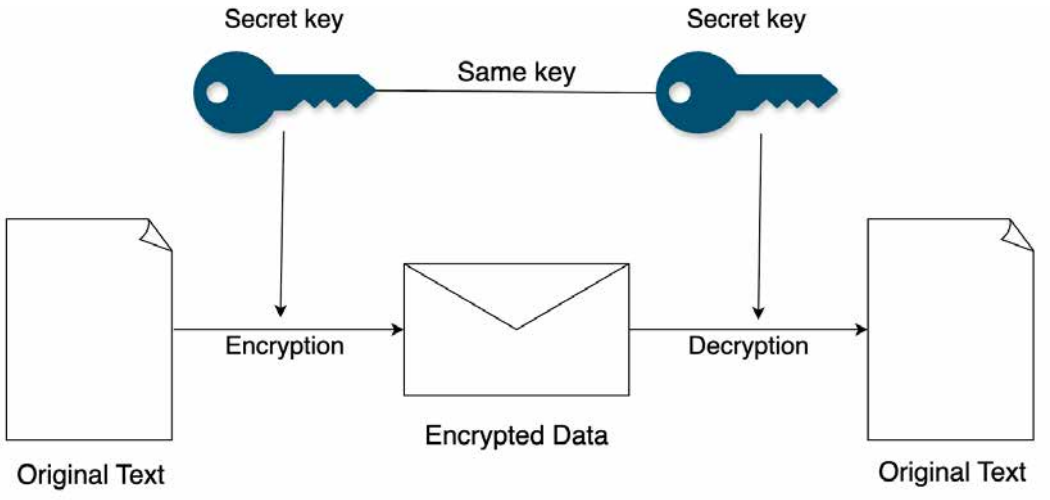

Рис. 2.1 Принцип дії симетричного шифрування.

Алгоритм AES – це сучасний блочний стандарт симетричного шифрування, затверджений Національним інститутом стандартів і технологій

США (NIST) у 2001 році як офіційна заміна DES [25]. Його створили бельгійські криптографи Вінсент Ріймен і Йоан Даймен у рамках відкритого конкурсу [4]. Завдяки поєднанню високої швидкості та стійкості до відомих атак AES став міжнародним стандартом безпеки (FIPS-197) і базовим механізмом для більшості сучасних фінансових систем.

AES працює з блоками даних фіксованого розміру 128 біт і підтримує ключі довжиною 128, 192 або 256 біт. Найбільш захищеною вважається модифікація AES-256, оскільки велика довжина ключа робить повний перебір практично неможливим навіть для потужних обчислювальних систем.

Процес шифрування складається з кількох послідовних кроків:

1. SubBytes – заміна байтів даних згідно з нелінійною таблицею перетворень (S-box).

2. ShiftRows – циклічне зсування рядків у матриці даних.

3. MixColumns – перемішування стовпців для підвищення дифузії.

4. AddRoundKey – комбінування даних із відповідним раундовим ключем.

Кількість раундів залежить від довжини ключа: 10 для AES-128, 12 для AES-192 і 14 для AES-256. Процес розшифрування виконує ці самі операції у зворотному порядку, що забезпечує симетричність перетворення [17].

Переваги AES-256:

- висока продуктивність як у програмних, так і в апаратних реалізаціях (CPU, HSM, смарт-карти);

- стійкість до криптоаналітичних атак – диференційних, лінійних, підстановкових та атак типу «відомий відкритий текст»;

- оптимальність для апаратного прискорення, зокрема у процесорах з підтримкою інструкцій AES-NI;

- відповідність міжнародним стандартам безпеки – FIPS-197, ISO/IEC 18033-3, PCI DSS, NIST SP 800-38.

У сучасних фінансових системах AES-256 використовується практично на всіх рівнях обробки даних:

- для зашифрування платіжних повідомлень у процесингових центрах і міжбанківських шлюзах (VisaNet, MasterCard, SWIFT);
- для захисту чутливої інформації у базах даних, включно з номерами карт, IBAN-кодами, CVV та персональними ідентифікаторами;
- у протоколах безпечного з'єднання між клієнтом і сервером (TLS, IPSec, VPN);
- для збереження резервних копій і токенізованих записів у хмарних сховищах.

Станом на 2025 рік AES-256 залишається одним із найнадійніших алгоритмів симетричного шифрування. За відсутності вразливостей у реалізації він практично не піддається зламу за допомогою класичних методів перебору. Єдиною потенційною загрозою для таких систем у майбутньому є розвиток квантових обчислень, які можуть зменшити ефективну криптостійкість деяких симетричних схем. Проте наразі AES-256 залишається основою для побудови безпечних фінансових сервісів і міжнародних стандартів захисту даних.

2. Асиметричні алгоритми шифрування

Асиметричне шифрування використовує пару математично пов'язаних ключів – відкритий (public) та закритий (private). Відкритий ключ доступний усім учасникам системи, тоді як закритий зберігається лише у власника. Якщо дані зашифровані відкритим ключем, їх можна розшифрувати лише за допомогою відповідного закритого ключа, і навпаки.

Цей підхід усуває проблему безпечного обміну ключами, яка існує в симетричних системах, але вимагає більшої обчислювальної потужності. Через це асиметричні алгоритми часто застосовують для захисту ключів симетричного шифрування, цифрового підпису та автентифікації. Приклад схеми роботи асиметричної криптосистеми показано на рисунку 2.2.

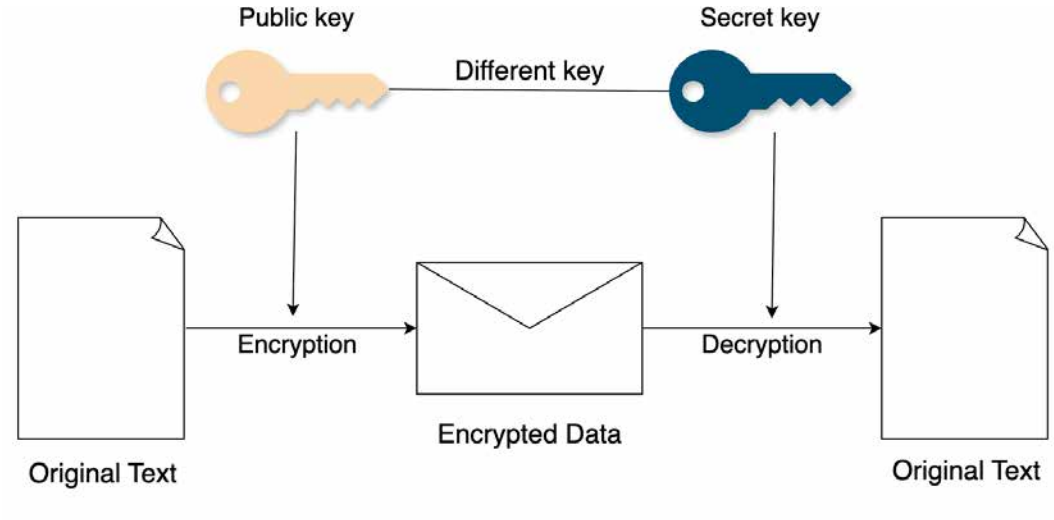

Рис. 2.2 Принцип дії асиметричного шифрування

У фінансових системах найпоширенішим стандартом залишається ECC, який використовується у протоколах безпечного обміну (TLS, HTTPS), цифрових сертифікатах і банківських токенах

Криптографія на еліптичних кривих (ECC) – це сучасний асиметричний метод захисту даних, у якому криптографічна стійкість ґрунтується не на факторизації великих чисел, як у RSA, а на задачі знаходження дискретного логарифма на еліптичних кривих. Ця задача вважається настільки складною, що навіть набагато коротші ключі забезпечують високий рівень безпеки. Саме тому ECC активно застосовується у фінансових сервісах, мобільних додатках, банківських протоколах та смарт-контрактах.

ECC використовує математичні властивості точок на еліптичних кривих. У системі генерується пара ключів:

- приватний ключ – випадкове число;
- публічний ключ – точка на кривій, отримана множенням генератора G на приватний ключ [13].

Операція множення на кривій проста й швидка, а ось обернена операція (тобто визначити приватний ключ за публічним) практично нездійсненна.

Процес шифрування:

1. Формування секретного ключа Генерується симетричний ключ (як правило – випадковий). Він використовується для шифрування даних.

2. Підготовка даних. Проводиться пакування, вирівнювання блоків (якщо блочний шифр), додавання метаданих, генерація вектора ініціалізації (IV) або nonce.

3. Шифрування даних. Реалізація залежить від типу алгоритму.

4. Генерація тегу автентичності.

5. Передавання зашифрованих даних.

6. Розшифрування відбувається у зворотному порядку.

Переваги ECC:

1. Висока криптостійкість при коротких ключах. Алгоритми на основі еліптичних кривих забезпечують дуже високий рівень захисту навіть при використанні компактних ключів. Це дозволяє зменшити обсяг даних, що передаються та зберігаються.

2. Низькі обчислювальні витрати. Операції над точками еліптичної кривої виконуються швидко й потребують менше обчислювальних ресурсів. Це робить ECC придатним для мобільних пристроїв, банківських терміналів та систем із високим навантаженням.

3. Компактні цифрові підписи та ключі. Короткі ключі ECC забезпечують можливість створення невеликих за розміром цифрових підписів, що позитивно впливає на швидкість обміну в захищених протоколах та зменшує трафік.

4. Висока ефективність у протоколах встановлення спільного секрету. Механізми на основі ECC (ECDH/ECDHE) дозволяють швидко формувати спільний симетричний ключ для безпечного сеансу, що особливо цінно у фінансових протоколах реального часу.

5. Сумісність із сучасними стандартами та інфраструктурами. ECC підтримується в SSL/TLS, мобільних платіжних системах, криптографічних токенах, блокчейні та цифрових підписах, що робить його універсальним рішенням для різних фінансових сервісів.

6. Енергоефективність. Завдяки меншій кількості обчислень ECC споживає менше енергії, що важливо для смарт-карт, токенів, POS-терміналів і пристроїв із обмеженими ресурсами.

7. Стійкість до сучасних атак на класичні криптосистеми. Математичні властивості еліптичних кривих забезпечують високий рівень стійкості до криптоаналітичних методів, що робить ECC надійним довгостроковим рішенням для захисту фінансових даних.

При проєктуванні систем криптографічного захисту фінансових операцій важливо враховувати не лише формальні властивості алгоритмів, а й їхню практичну придатність у контексті конкретних навантажень, середовищ та обмежень. У сучасних фінансових протоколах ключову роль відіграють дві асиметричні криптосистеми – RSA та криптографія на еліптичних кривих (ECC). Обидві використовуються для автентифікації, цифрового підпису та безпечного обміну ключами, проте значно різняться за швидкодією, ресурсними потребами та рівнем безпеки при однаковій довжині ключів. Для систем, де затримка має критичне значення (мобільний банкінг, платіжні шлюзи, сервіси з високою інтенсивністю транзакцій), вибір між RSA та ECC визначає не лише швидкість обробки запитів, а й можливість масштабування всієї інфраструктури. У таблиці 2.3 наведено порівняльну характеристику RSA та ECC за головними параметрами, які мають ключове значення для фінансових систем.

Таблиця 2.3

Порівняння RSA та ECC за основними параметрами

Параметр	RSA-2048	RSA-4096	ECC-256	ECC-521
Рівень безпеки	Середній	Високий	Високий	Дуже високий
Швидкодія	Низька	Дуже низька	Середня/висока	Середня
Операції підпису	Повільні	Дуже повільні	Швидкі	Швидкі
Вимоги до пам’яті	Високі	Дуже високі	Низькі	Середні
Рекомендоване використання	Сервери, SSL	Архіви, HSM	Мобільні платежі, IoT	Уряди, міжнародні платежі

Порівняння параметрів RSA та ECC показує, що криптографія на еліптичних кривих демонструє значно кращу ефективність за співвідношенням

«рівень безпеки – обчислювальні витрати». ECC забезпечує еквівалентну або вищу криптостійкість при ключах, які у десятки разів коротші за RSA, що істотно зменшує навантаження на процесор і пам’ять. Це робить ECC придатним для мобільних пристроїв, вбудованих систем та високонавантажених платіжних сервісів, де затримка криптографічних операцій безпосередньо впливає на зручність користувача та пропускну здатність системи. Натомість RSA, попри свою надійність і багаторічне поширення, стає менш ефективним через великі ключі та високий час виконання операцій підпису та перевірки, особливо у конфігураціях з обмеженими ресурсами або при великій кількості паралельних підключень. У результаті аналізу можна зробити висновок, що для більшості сучасних фінансових сервісів доцільним є поступовий перехід до ECC або гібридних схем, де ECC відповідає за обмін ключами, а симетричні алгоритми – за основне шифрування даних.

Для повнішого розуміння практичних відмінностей між алгоритмами RSA та ECC доцільно проаналізувати їхню продуктивність під час виконання базових криптоопераций – формування цифрового підпису та перевірки його достовірності. Саме ці операції визначають затримку під час встановлення захищених сеансів та здійснення фінансових транзакцій у режимі реального часу. На рисунку 2.3 наведено узагальнену оцінку часу виконання криптоопераций для найпоширеніших конфігурацій RSA та ECC.

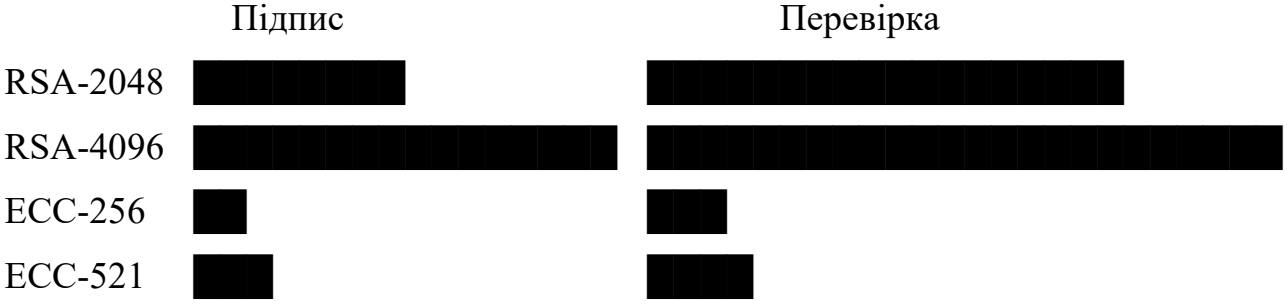

Рис. 2.3 Порівняння часу підпису/перевірки для RSA та ECC

Представлений графік демонструє істотну різницю між сімействами алгоритмів за обчислювальною складністю криптоопераций. Алгоритм RSA, особливо у конфігурації 4096 біт, потребує значних ресурсів як для підпису, так

і для перевірки, що зумовлює збільшення затримки при обробці транзакцій. Натомість алгоритми ECC забезпечують суттєво менший час виконання операцій, що особливо важливо для високонавантажених платіжних систем та мобільних застосунків, де кожна мілісекунда впливає на загальну швидкість обслуговування користувачів. Це підтверджує доцільність використання ECC у сучасній фінансовій криптографії, де потрібний баланс між безпекою та продуктивністю.

2.5 Порівняльний аналіз симетричних, асиметричних та гібридних алгоритмів шифрування

У фінансових системах алгоритми на еліптичних кривих (ECC) найчастіше застосовуються разом із AES-256 у складі гібридних криптографічних протоколів. У такій моделі ECC відповідає за безпечний обмін симетричними ключами та автентифікацію сторін, тоді як AES використовується для швидкого шифрування власне транзакційних даних [6; 25]. Такий підхід реалізовано в сучасних версіях протоколів TLS/SSL, у VPN-технологіях, цифрових сертифікатах та інфраструктурах відкритих ключів (PKI), де ECC поступово замінює традиційні асиметричні системи завдяки своїй високій криптостійкості та кращій продуктивності.

Симетричні алгоритми шифрування, такі як AES базуються на використанні одного спільного ключа для шифрування і розшифрування даних. Вони характеризуються високою швидкодією та ефективністю при роботі з великими обсягами транзакцій.

Переваги симетричних систем:

- висока продуктивність: симетричне шифрування виконується у десятки разів швидше, ніж асиметричне, що є критично важливим для банківських процесингових систем;

- невелике навантаження на апаратні ресурси: алгоритми типу AES оптимізовані для реалізації як у програмному, так і в апаратному середовищі [20];

- надійна криптографічна стійкість: зокрема, AES-256 офіційно затверджений стандартом FIPS-197 і на сьогодні вважається практично незламним при правильному впровадженні.

Недоліки симетричних систем:

- проблема безпечного обміну ключами: необхідність передачі секретного ключа між сторонами створює ризик його перехоплення;

- відсутність механізму автентифікації: Неможливо достовірно встановити, хто саме створив повідомлення, оскільки обидві сторони використовують один ключ;

- складність масштабування: у великих мережах кількість необхідних ключів швидко зростає, що ускладнює адміністрування.

Асиметричні алгоритми – такі як RSA (Rivest–Shamir–Adleman), ECC (Elliptic Curve Cryptography) та DSA (Digital Signature Algorithm) – використовують пару ключів: відкритий для шифрування й закритий для розшифрування. Це вирішує проблему розповсюдження ключів, однак створює додаткове навантаження на систему.

Переваги асиметричних систем:

- безпечний обмін ключами: відкритий ключ може передаватися без загрози компрометації системи;

- можливість створення цифрового підпису: асиметричні алгоритми забезпечують невідмовність і юридичну значущість транзакцій;

- сумісність із інфраструктурою PKI (Public Key Infrastructure): використовується для сертифікації користувачів, серверів і пристроїв.

Недоліки асиметричних систем:

- низька швидкодія: RSA-2048 працює значно повільніше за AES-256, тому не придатний для масового шифрування великих потоків даних;

- високі вимоги до обчислювальних ресурсів: довгі ключі (2048–4096 біт) потребують значних потужностей процесора;

- потенційна вразливість до квантових атак: розвиток квантових обчислень може у майбутньому поставити під загрозу алгоритми, що базуються на факторизації чисел.

Гібридні криптосистеми поєднують симетричне й асиметричне шифрування, об'єднуючи їхні переваги. У таких схемах RSA або ECC використовуються для захищеної передачі симетричного ключа, а AES- 256 – для шифрування самих даних.

Цей підхід є базовим для протоколів TLS/SSL, SWIFT, VisaNet, MasterCard Network, а також систем інтернет-банкінгу.

Переваги гібридних систем:

- висока ефективність: шифрування даних відбувається швидко, а ключі обмінюються безпечно;

- гнучкість: можливість адаптації до різних рівнів безпеки та обчислювальних ресурсів;

- масштабованість: добре підходять для складних розподілених систем і хмарних платформ.

Недоліки гібридних систем:

- складність реалізації: потрібна інтеграція кількох криптографічних модулів і правильна організація управління ключами;

- залежність від налаштувань: помилка в конфігурації протоколу (наприклад, SSLv3 або слабкий хеш) може призвести до компрометації.

Гібридні криптографічні протоколи поєднують переваги симетричних і асиметричних алгоритмів, забезпечуючи високу швидкодію шифрування та стійкий механізм обміну ключами. Саме вони лежать в основі сучасних платіжних стандартів і мережевих протоколів, де одночасно необхідні мінімальна затримка та максимальна криптостійкість. У фінансових системах гібридні схеми застосовуються в захищених платіжних каналах, автентифікації терміналів, створенні зашифрованих сесій мобільних застосунків та електронних

сервісів, що вимагають оперативного встановлення безпечних сесій. У таблиці 2.4 наведено порівняння основних гібридних криптографічних протоколів, що використовуються у високозахищених інформаційних середовищах.

Таблиця 2.4

Гібридні криптографічні протоколи

Протокол	Симетрична частина	Асиметрична частина	Стійкість	Особливості
TLS 1.3	AES-256, ChaCha20	ECC (X25519)	Дуже висока	Forward secrecy, короткий handshake
OpenPGP	AES, Twofish	RSA/ECC	Висока	Автономна модель ключів
SSH	AES, ChaCha20	RSA/ECDSA	Висока	Використання Ed25519 за замовчуванням

Аналіз гібридних криптографічних протоколів свідчить, що найбільшу ефективність демонструють системи, які використовують асиметричні алгоритми лише на початковому етапі встановлення сеансу, а подальший обмін даними здійснюють за допомогою високошвидкісних симетричних шифрів. Такий підхід забезпечує як криптостійкість, так і продуктивність, що є критично важливим у середовищах із високою частотою транзакцій, наприклад, у банківських мобільних застосунках та процесингових системах.

Протоколи TLS 1.3 та SSH демонструють значну перевагу завдяки використанню еліптичних кривих для швидкого встановлення ключа та ChaCha20 або AES-256 для подальшого шифрування трафіку. У підсумку саме гібридні схеми створюють оптимальний баланс між захистом та швидкодією, що пояснює їх домінування у фінансовій галузі [25].

Для глибшого розуміння принципів роботи гібридних протоколів важливо проаналізувати їхню алгоритмічну складність. Відмінності між симетричною та асиметричною частинами прямо впливають на затримку встановлення захищеного каналу, а також на загальну швидкість подальшої передачі даних. На рисунку 2.4 подано узагальнену модель складності гібридної криптосхеми, яка

показує співвідношення часових витрат між симетричною частиною, асиметричною фазою обміну ключами та загальними витратами протоколу.

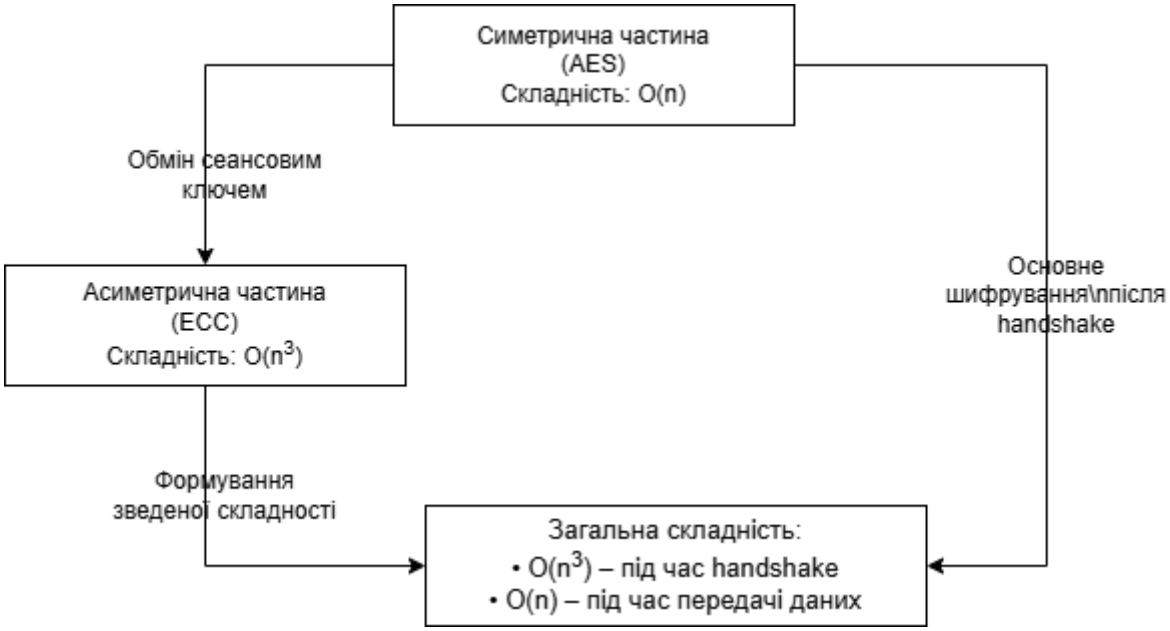

Рис. 2.4 Складність гібридної криптосхеми

Представлена схема демонструє, що найбільш ресурсомісткою частиною гібридного протоколу є асиметричний етап формування ключа, складність якого наближено оцінюють як $O(n^3)$. Саме він визначає базові витрати під час handshake та формує основну затримку при встановленні сесії. Натомість симетричне шифрування має лінійну складність $O(n)$, що забезпечує високу швидкість передачі даних після завершення початкового обміну ключами. Таким чином, схема підтверджує, що ефективність гібридних протоколів базується на одноразовому виконанні дорогих операцій з асиметричними ключами та багаторазовому використанні швидких симетричних алгоритмів для подальшого захисту фінансових трансакцій. Це дозволяє одночасно досягти високої пропускної здатності та стійкості до криптоаналітичних атак.

У фінансових інформаційних системах криптографічні алгоритми застосовуються для розв’язання різних завдань – від шифрування даних і формування цифрових підписів до автентифікації клієнтів та обміну ключами. Тому важливо не лише розглядати окремі алгоритми в межах певних груп, а й проводити їх узагальнене порівняння за ключовими характеристиками, які

визначають раціональність вибору криптографічних механізмів для конкретної інфраструктури. Така сумарна характеристика дозволяє встановити оптимальний баланс між швидкодією, рівнем безпеки, вимогами до пам’яті та сферами застосування, що є критично важливим при проєктуванні систем захисту фінансових операцій. У таблиці 2.5 наведено порівняльний аналіз найпоширеніших алгоритмів симетричної та асиметричної криптографії, що широко використовуються у сучасних платіжних сервісах та банківських протоколах.

Таблиця 2.5

Порівняльний аналіз AES, RSA, ECC, ChaCha20 і Twofish

Алгоритм	Тип	Швидкодія	Рівень безпеки	Пам’ять	Оптимальний сценарій
AES-256	Симетричний	Висока	Дуже висока	Низька	Банківські транзакції, TLS
ChaCha20	Симетричний	Дуже висока	Висока	Дуже низька	Мобільні платежі
Twofish	Симетричний	Середня	Висока	Середня	Архівування, офлайн-системи
RSA-2048	Асиметричний	Низька	Середня	Висока	Сертифікати, автентифікація
ECC-256	Асиметричний	Середня/висока	Висока	Низька	Смарт-карти, мобільний банкінг

Аналіз порівняльних характеристик засвідчує, що кожен із розглянутих алгоритмів виконує чітко визначену функцію у криптографічній інфраструктурі. AES та ChaCha20 забезпечують найвищу продуктивність під час шифрування значних обсягів даних, що робить їх основою для захисту каналу зв’язку та транзакцій у режимі реального часу. Twofish демонструє високу криптостійкість, проте поступається лідерам за швидкодією, тому найчастіше використовується у системах з помірним навантаженням або для довготривалого зберігання зашифрованих даних. RSA та ECC мають зовсім інше призначення – автентифікацію та цифрові підписи, але ECC переважає RSA за співвідношенням безпеки та ресурсних витрат, що підтверджує актуальність його використання у мобільних та високонавантажених фінансових застосунках. Таким чином, оптимальна система захисту фінансових операцій повинна

базуватися на поєднанні різних алгоритмів, а не на одному універсальному рішенні.

З метою поглибленого порівняння ключових криптографічних алгоритмів доцільно візуалізувати їх співвідношення за основними параметрами – швидкодією, рівнем криптостійкості та вимогами до пам’яті. Така графічна форма подання дозволяє наочно оцінити відмінності між алгоритмами, їхні сильні сторони та потенційні обмеження при інтеграції у реальні фінансові системи. На рисунку 2.5 наведено порівняльну діаграму, що узагальнює ключові властивості AES, ChaCha20, Twofish, RSA та ECC у компактній і структурованій формі.

Рис. 2.5 Порівняння криптографічних алгоритмів за ключовими параметрами

Діаграма відображає суттєві відмінності між алгоритмами з точки зору їх придатності для практичного застосування у фінансових системах. AES-256 та ChaCha20 демонструють найвищу швидкодію, що робить їх оптимальними для шифрування великих потоків даних у реальному часі. ECC-256 забезпечує високу криптостійкість при мінімальних вимогах до пам’яті, тоді як RSA-2048, попри свою надійність, має значні ресурсні витрати, що обмежує його застосування у високопродуктивних середовищах. Twofish займає проміжну позицію, пропонуючи високу безпеку за рахунок дещо нижчої швидкодії. Загалом представлена візуалізація підтверджує, що вибір криптографічного алгоритму повинен базуватися на конкретних технічних умовах, обсязі даних та вимогах до швидкості обробки транзакцій.

Отже, гібридна модель AES + ECC вважається найефективнішим рішенням для сучасних банківських систем, забезпечуючи баланс між безпекою, продуктивністю та юридичною достовірністю транзакцій.

РОЗДІЛ 3. МОДЕЛЮВАННЯ СИСТЕМИ КРИПТОГРАФІЧНОГО ЗАХИСТУ

3.1 Поняття, мета та завдання моделювання систем захисту

Моделювання інформаційних систем – це процес створення формалізованого опису структури, поведінки та взаємодії компонентів системи для їх подальшого аналізу, проєктування та реалізації. У контексті системи криптографічного захисту фінансових операцій моделювання дозволяє визначити логіку обробки даних, описати основні бізнес-процеси, виявити взаємозв’язки між підсистемами та оцінити ефективність архітектурних рішень до початку розробки.

Метою моделювання є створення цілісного уявлення про систему, що забезпечує:

- структурованість і логічну узгодженість її компонентів;
- оптимізацію процесів шифрування, автентифікації та передавання даних;
- можливість оцінки потенційних загроз і слабких місць до реалізації системи;
- спрощення подальшого етапу розробки завдяки формалізованим описам.

Основними завданнями моделювання криптографічної системи є:

- опис функціональних можливостей системи з позиції користувачів і розробників;
- визначення об’єктів, які взаємодіють між собою (користувач, сервер, криптографічний модуль, база даних);
- побудова моделей поведінки системи під час обробки фінансових транзакцій;
- створення UML-діаграм, що відображають функціональні, динамічні та структурні аспекти системи.

3.2 Функціональна модель системи криптографічного захисту

На рис. 3.1 зображена функціональна модель системи криптографічного захисту фінансових операцій, яка побудована у нотації IDEF0 та відображає логічну структуру процесів. Вони забезпечують цілісність, конфіденційність і достовірність обробки транзакцій у банківській або фінтех-системі.

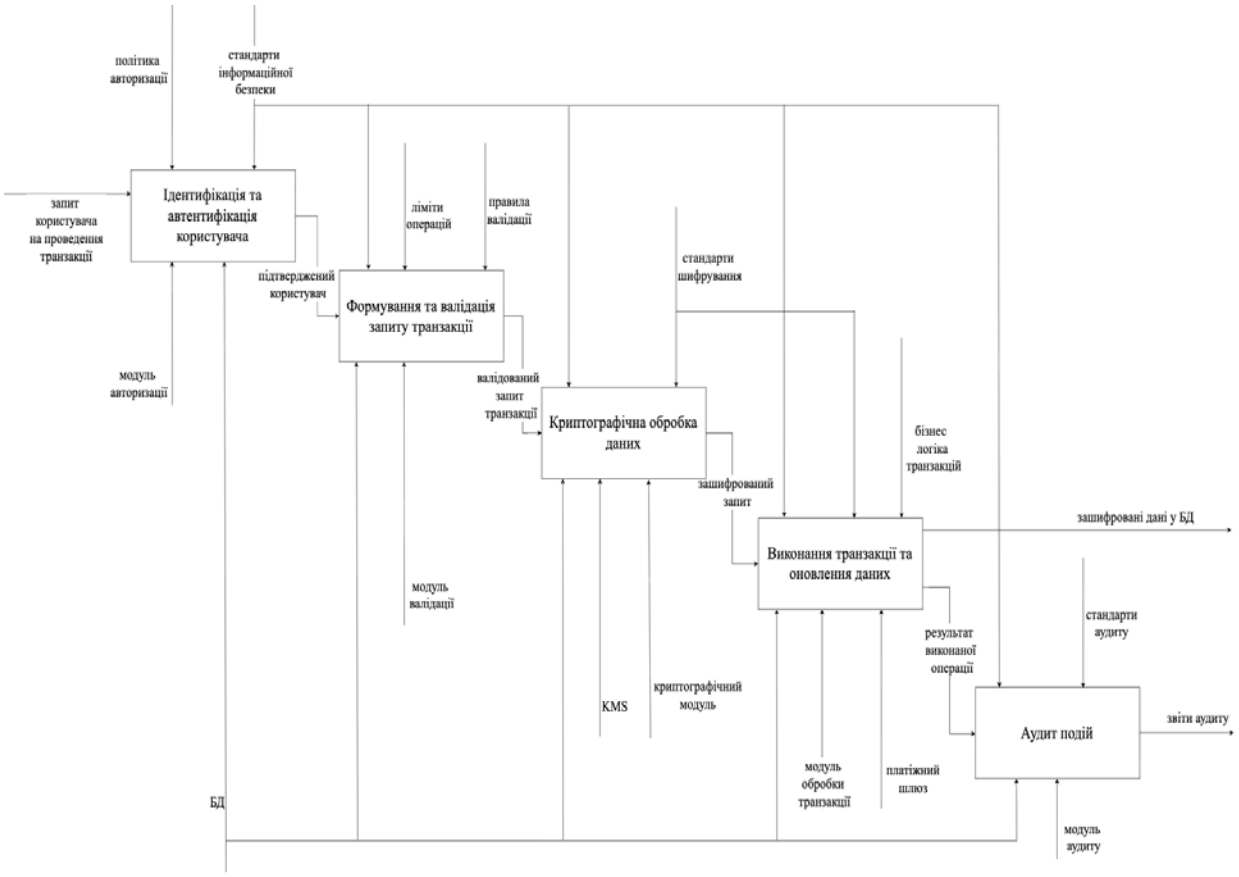

Рис.3.1 Функціональна модель системи.

Система складається з п’яти основних функцій:

- A1 – Ідентифікація та автентифікація користувача;
- A2 – Формування та валідація запиту транзакції;
- A3 – Криптографічна обробка даних;
- A4 – Виконання транзакції та оновлення даних;
- A5 – Аудит подій.

Вхідними даними моделі є запит користувача на проведення транзакції, який надходить через клієнтський інтерфейс системи.

Результатом роботи є зашифровані дані, що зберігаються в базі даних, а також звіти аудиту, які використовуються для подальшого аналізу безпеки та відповідності нормативним вимогам.

Опис основних функцій:

1. Ідентифікація та автентифікація користувача.

На цьому етапі система отримує запит користувача з його обліковими даними (логін, пароль, токен або сертифікат). Процес контролюється політикою авторизації та стандартами інформаційної безпеки, що визначають правила перевірки особи, кількість допустимих спроб входу, вимоги до складності пароля тощо. Для реалізації функції використовується модуль авторизації, який звертається до бази даних користувачів. Результатом є підтверджений користувач, який отримує дозвіл на формування запиту транзакції.

2. Формування та валідація запиту транзакції.

Після успішної автентифікації користувач формує запит на проведення фінансової операції. Система перевіряє введені дані на коректність і відповідність правилам валідації та лімітам операцій (максимальна сума, дозволений тип транзакції, наявність коштів на рахунку). Процес виконується за допомогою модуля валідації, який отримує допоміжні дані з бази даних. Результатом є валідований запит транзакції, що передається до криптографічного модуля для подальшої обробки.

3. Криптографічна обробка даних.

Ця функція відповідає за захист усіх фінансових даних, які проходять через систему. Процес здійснюється відповідно до стандартів шифрування та політик безпеки (AES-256, ECC, TLS 1.3). Використовується криптографічний модуль та сервіс управління ключами (KMS), який забезпечує генерацію, ротацію та зберігання криптографічних ключів. Результатом цього етапу є зашифрований запит транзакції, який гарантує захист даних від несанкціонованого доступу під час передавання чи зберігання.

4. Виконання транзакції та оновлення даних.

Отримавши зашифрований запит, система розшифровує його за допомогою ключів із KMS і виконує фінансову операцію згідно з бізнес-логікою транзакцій. Цей процес регулюється стандартами інформаційної безпеки та внутрішніми регламентами контролю цілісності даних. Для реалізації функції застосовується модуль обробки транзакцій, який взаємодіє з платіжним шлюзом і базою даних. Результатом є зашифровані дані у базі даних та результат виконаної операції, що передається далі для аудиту.

5. Аудит подій.

Функція A5 забезпечує контроль, моніторинг та аналітичний облік усіх дій у системі, пов’язаних із проведенням фінансових транзакцій та обробкою конфіденційних даних. Метою аудиту є виявлення потенційних інцидентів безпеки, забезпечення прозорості дій користувачів і адміністраторів, а також підтвердження цілісності журналів для подальшої перевірки відповідності системи міжнародним стандартам (зокрема ISO/IEC 27001, PCI DSS, NIST SP 800-53) [15; 21].

3.3 Об’єктно-орієнтована модель системи

Об’єктно-орієнтована модель системи криптографічного захисту фінансових операцій відображає основні класи, їхні атрибути, методи та взаємозв’язки між ними. Модель побудована на основі принципів інкапсуляції, модульності та повторного використання коду, що забезпечує гнучкість і розширюваність системи. На рис. 3.2 представлено UML-діаграму класів, яка описує логічну структуру компонентів системи.

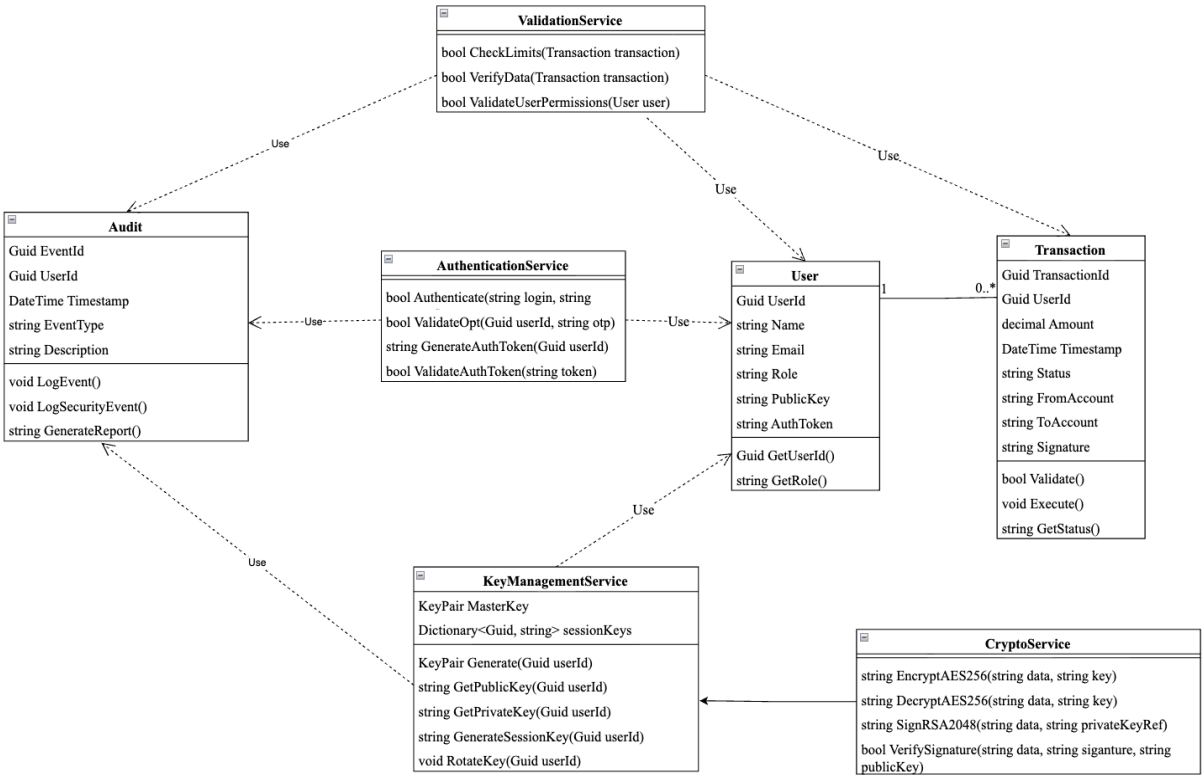

Рис. 3.2 UML-діаграма системи.

Основні класи системи:

1. User – клас, що представляє зареєстрованого користувача системи. Містить ідентифікатор користувача, ім’я, електронну пошту, роль у системі, публічний ключ та токен автентифікації. Клас забезпечує методи GetUserId() та GetRole(), які використовуються іншими модулями для авторизації та контролю доступу. Один користувач може мати декілька транзакцій.

2. Transaction –клас для представлення фінансової операції. Містить ідентифікатор транзакції, ідентифікатор користувача, суму переказу, валюту, дату та час створення, статус виконання, а також реквізити відправника і одержувача. Передбачає методи перевірки коректності (Validate()), виконання (Execute()) і отримання поточного стану (GetStatus()). Транзакція проходить етапи перевірки, шифрування та підпису перед відправленням на виконання.

3. Сервіс автентифікації – компонент, що забезпечує вхід користувачів у систему. Реалізує перевірку облікових даних, валідацію одноразових кодів (OTP), створення та перевірку токенів автентифікації. Всі події автентифікації реєструються в журналі безпеки для подальшого аудиту.

4. Сервіс валідації – модуль, що перевіряє правильність даних транзакції перед виконанням. Контролює дотримання фінансових лімітів, відповідність бізнес-правилам і рівням доступу користувача. Результати перевірок зберігаються у журналі подій безпеки.

5. Сервіс управління ключами – компонент, який відповідає за генерацію, зберігання, оновлення та ротацію криптографічних ключів. Використовує головний ключ і набір сесійних ключів для різних користувачів. Надає інтерфейси для отримання публічного та приватного ключів, а також створення нових сеансових ключів. Всі дії з ключами реєструються в системі аудиту.

6. Криптографічний сервіс – модуль, що виконує криптографічні операції. Реалізує симетричне шифрування і розшифрування даних за допомогою алгоритму AES-256, а також створення і перевірку електронного підпису на основі ECC. Для доступу до ключів використовує сервіс управління ключами.

7. Модуль аудиту (Audit) – компонент системи, який відповідає за журналювання подій. Зберігає інформацію про автентифікацію користувачів, виконання транзакцій, перевірки даних і криптографічні операції. Містить ідентифікатор події, користувача, тип події, час і короткий опис. Забезпечує ведення журналу безпеки та формування звітів для моніторингу стану системи.

3.4 Взаємодія модулів у процесі виконання фінансової транзакції

Діаграма послідовності описує порядок взаємодії між основними компонентами системи криптографічного захисту фінансових операцій під час виконання транзакції. Вона демонструє, як користувач ініціює процес, а система

послідовно забезпечує перевірку, шифрування, виконання та підтвердження операції.

На рисунку 3.3 зображено основні об’єкти: користувач, клієнтський інтерфейс, серверна сторона, криптографічний сервіс та база даних. Кожен з них виконує власну роль у процесі обробки транзакції та взаємодіє через обмін повідомленнями.

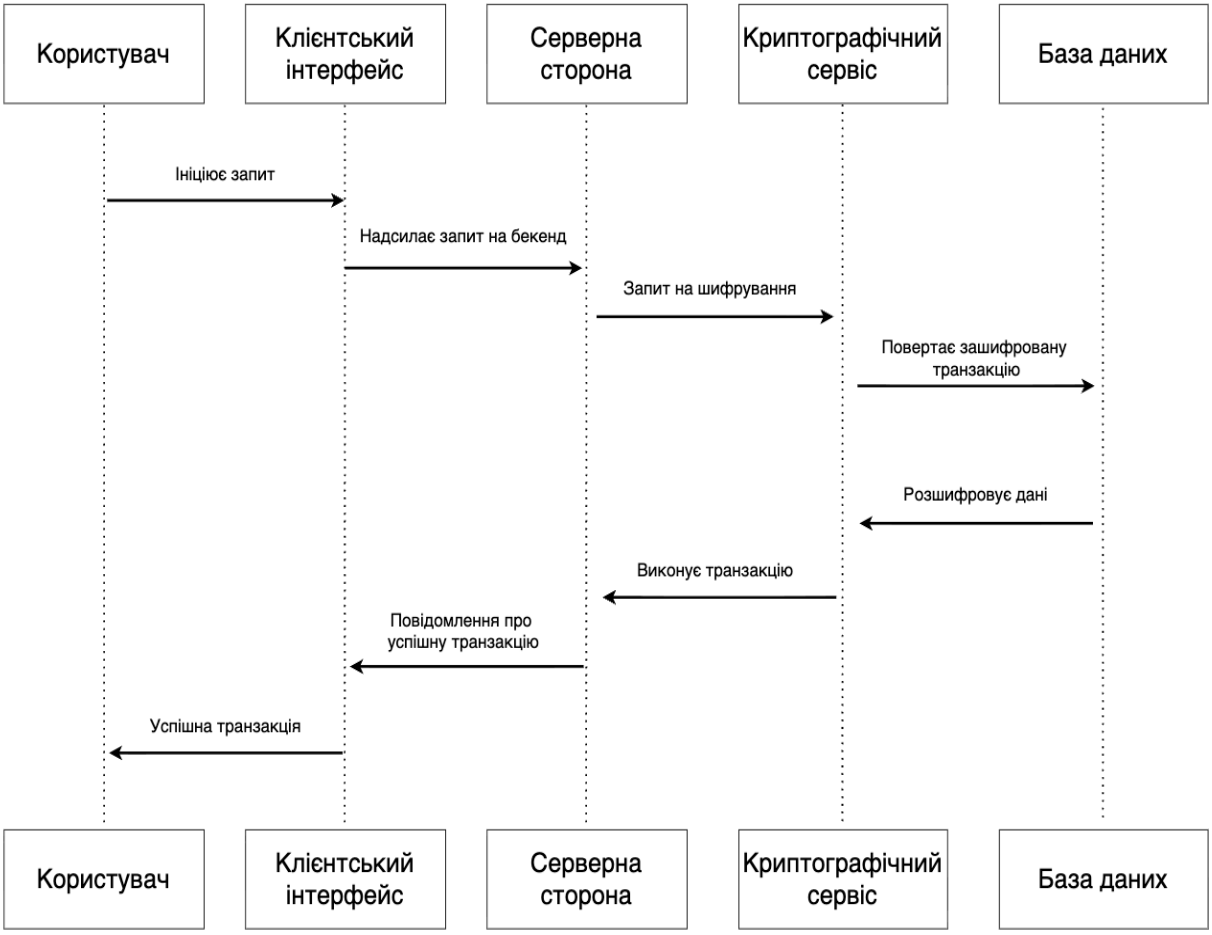

Рис.3.3 Діаграма послідовності у процесі виконання фінансової транзакції.

Послідовність подій починається з того, що користувач ініціює запит на проведення фінансової операції через клієнтський інтерфейс. Після цього клієнтська частина системи формує запит і передає його на сервер для подальшої обробки. Серверна сторона отримує запит, проводить попередню валідацію даних і звертається до криптографічного сервісу для виконання шифрування фінансової інформації (використовуючи алгоритм AES-256).

Отримавши зашифровані дані, сервер виконує фінансову транзакцію, взаємодіючи з базою даних, у якій результати операції зберігаються у зашифрованому вигляді. Після завершення операції система передає зашифровані результати назад до криптографічного сервісу, де відбувається розшифрування даних для формування повідомлення про результат.

Далі сервер надсилає повідомлення про успішну транзакцію клієнтському інтерфейсу, який, у свою чергу, інформує користувача про завершення операції. Таким чином, діаграма демонструє замкнутий цикл обробки транзакції – від ініціації запиту до повернення підтвердження користувачу.

РОЗДІЛ 4. РОЗРОБКА КРИПТОГРАФІЧНОЇ СИСТЕМИ ЗАХИСТУ ФІНАНСОВИХ ОПЕРАЦІЙ

4.1 Технології, середовище розробки та реалізація компонентів

Розроблення системи криптографічного захисту фінансових операцій здійснювалось із використанням сучасних технологій, що забезпечують високу надійність, безпеку та масштабованість програмного рішення. Архітектура системи передбачає розподіл компонентів на клієнтську, серверну та криптографічну частини, які взаємодіють між собою через захищені канали зв’язку.

Основна логіка системи реалізована мовою C# у середовищі .NET 8.0, що забезпечує ефективне управління пам’яттю, підтримку асинхронних операцій та вбудовані засоби безпеки. Для зберігання даних використано PostgreSQL, який підтримує надійне зберігання великих обсягів фінансової інформації та можливість роботи із зашифрованими полями. Комунікація між клієнтською та серверною частинами реалізована через REST API з використанням протоколу HTTPS і стандарту TLS 1.3 для шифрування транспортного рівня.

Модуль криптографічного захисту побудований на основі бібліотек System.Security.Cryptography, які реалізують симетричні та асиметричні алгоритми шифрування.

1. Для шифрування даних застосовується алгоритм AES-256 у режимі GCM, який забезпечує як конфіденційність, так і перевірку цілісності повідомлень.

2. Для підпису транзакцій використовується алгоритм ECC, що гарантує автентичність операцій.

3. Обмін ключами виконується за допомогою KMS, який відповідає за генерацію, зберігання та ротацію криптографічних ключів.

Клієнтська частина реалізована як веб-інтерфейс на базі React.js, який забезпечує інтуїтивно зрозумілий інтерфейс користувача та взаємодію із сервером через REST API.

На сервері впроваджено AuthenticationService, який забезпечує авторизацію користувачів, а також ValidationService, що виконує логічну перевірку транзакцій перед їх виконанням. Усі події в системі логуються за допомогою модуля Audit, що зберігає інформацію про дії користувачів, спроби автентифікації та результати виконаних операцій.

Розгортання серверної частини відбувається у Docker-контейнерах, що забезпечує ізоляцію середовища виконання, повторюваність конфігурацій і простоту масштабування. Завдяки цьому система може бути інтегрована у фінансову інфраструктуру з мінімальними витратами на адаптацію.

4.2 Архітектура системи та її компоненти

Архітектура системи криптографічного захисту фінансових операцій має модульну трирівневу структуру, яка забезпечує розподіл функціональності між клієнтською частиною, серверною частиною та базою даних. На рис. 4.1 представлено загальну архітектуру розроблюваної системи.

1. Клієнтська частина.

Клієнтський рівень є точкою взаємодії користувача з системою. Основним компонентом клієнтської частини є модуль шифрування (Encryption Module), який виконує попереднє шифрування фінансових даних перед їх передачею на сервер. Для цього використовується симетричний алгоритм AES-256, який забезпечує високий рівень криптостійкості при мінімальних затратах обчислювальних ресурсів. Передача зашифрованих даних до серверної частини відбувається через захищений канал HTTPS з використанням протоколу TLS 1.3, що гарантує цілісність і конфіденційність даних на транспортному рівні.

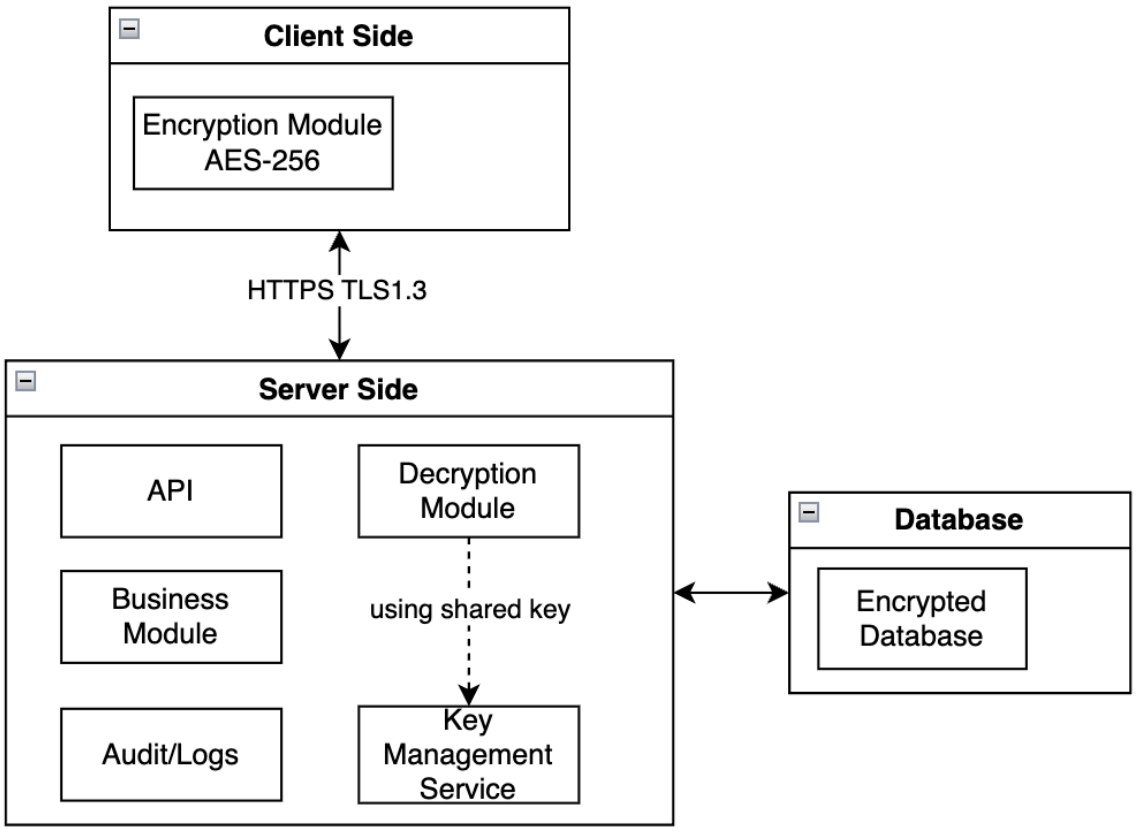

Рис.4.1 Архітектура системи.

2. Серверна частина.

Серверний рівень виконує основну бізнес-логіку системи, обробку запитів користувачів, валідацію, розшифрування даних і формування результатів транзакцій. Вона складається з кількох взаємопов’язаних модулів:

- API-модуль – забезпечує взаємодію між клієнтською частиною і сервером, приймає запити користувачів, передає їх на обробку і повертає результати у стандартизованому форматі JSON. Також відповідає за авторизацію та контроль доступу.

- Модуль бізнес-логіки (Business Module) – реалізує основні сценарії виконання фінансових операцій. Він координує процеси валідації, звернення до криптографічного модуля, оновлення даних у базі та створення журналів подій.

- Модуль дешифрування (Decryption Module) – виконує розшифрування вхідних даних, отриманих від клієнтської частини, за допомогою симетричного

ключа, яким керує служба управління ключами (KMS).Забезпечує цілісність даних і перевіряє автентичність повідомлення перед передачею в бізнес-модуль.

- Служба управління ключами (Key Management Service) – відповідає за генерацію, зберігання та ротацію криптографічних ключів, необхідних для шифрування й дешифрування. Використовує комбінацію симетричних (AES) і асиметричних (ECC) ключів. Обмін ключами здійснюється у зашифрованому вигляді, що запобігає компрометації при передачі.

- Модуль аудиту (Audit/Logs) – логує інформацію про всі події системи: автентифікацію користувачів, виконання транзакцій, криптографічні операції. Це дозволяє відстежувати дії користувачів, проводити аналіз інцидентів безпеки та забезпечувати відповідність стандартам фінансової звітності (наприклад, PCI DSS).

3. База даних

База даних використовується для зберігання інформації про користувачів, фінансові операції, транзакційні журнали та службові події системи. У рамках концепції диференційованого захисту даних шифруванню підлягають лише чутливі записи, що містять інформацію про платіжні картки, реквізити або інші персональні фінансові дані. Такі поля зберігаються в зашифрованому вигляді з використанням алгоритму AES- 256 на рівні застосунку (*application-level encryption*), тобто шифрування виконується до потрапляння даних у базу.

Решта інформації зберігається у відкритому вигляді, але в межах захищеного середовища. Цей підхід дозволяє знизити навантаження на систему шифрування, зберігаючи при цьому високий рівень безпеки для критичних даних.

Крім того, сама база даних захищена на верхньому рівні – доступ до неї здійснюється виключно через авторизований серверний модуль, який працює у середовищі з TLS-з’єднанням та апаратним шифруванням на рівні диску (disk-level encryption). Це означає, що навіть у разі несанкціонованого доступу до фізичного сховища, зловмисник не зможе відновити вміст таблиць без відповідних ключів шифрування.

4.3 Протоколи безпечної взаємодії

У процесі розробки криптографічної системи захисту ключове значення має вибір протоколів, які забезпечують безпечну передачу даних, захист від перехоплення та підробки інформації, а також підтвердження автентичності учасників фінансової транзакції. Сучасна фінансова інфраструктура ґрунтується на поєднанні кількох взаємодоповнюючих протоколів, кожен з яких виконує власну функцію в межах єдиного механізму безпеки. Вони формують багаторівневу систему захисту, що гарантує конфіденційність, цілісність та достовірність переданих даних, а також стійкість до атак «людина посередині», підміни сертифікатів і несанкціонованого доступу.

Одним із ключових компонентів системи є протокол TLS 1.3, що використовується для створення захищеного каналу зв’язку між клієнтом і сервером API. На відміну від попередніх версій TLS, нова специфікація користується одноетапним механізмом узгодження параметрів, що суттєво зменшує затримку під час встановлення сесії. У процесі handshake застосовуються еліптичні криві, переважно X25519, які забезпечують швидкий і водночас криптостійкий обмін ключами. Усі передані дані шифруються симетричними алгоритмами AES-256, що дозволяє зберігати високу продуктивність навіть у сценаріях з інтенсивним трафіком. Важливою властивістю TLS 1.3 є forward secrecy, що гарантує неможливість розшифрування минулих сесій навіть у випадку компрометації довгострокового ключа сервера.

У межах системи TLS виступає базовим шаром безпеки для протоколу HTTPS, який використовується як транспортний механізм для REST-запитів. HTTPS забезпечує повне шифрування запиту й відповіді, автентифікацію сервера шляхом перевірки його сертифіката й унеможливлює перехоплення або модифікацію даних під час передачі. Таким чином, використання HTTPS створює захищений канал для бізнес-логіки API,

дозволяючи клієнту достовірно встановити, що він взаємодіє саме з легітимним сервером, а не з підміненою стороною.

У другому захисному шарі застосовується механізм авторизації JWT, який дозволяє забезпечити контроль доступу до ресурсів системи. JWT-токен підписується за допомогою алгоритму ES256 на основі еліптичних кривих, що забезпечує високу стійкість підпису при мінімальних обчислювальних витратах. Усередині токена містяться ідентифікаційні дані користувача, часові межі його дії та службова інформація, необхідна для перевірки прав доступу. Завдяки криптографічному підпису сервер може перевірити достовірність токена локально, без звернення до бази даних, що зменшує навантаження на систему й пришвидшує обробку транзакцій. Такий підхід особливо важливий у сценаріях високої інтенсивності запитів, характерних для мобільних банківських застосунків та електронних платіжних сервісів.

Ключовим елементом перевірки достовірності інформації є цифровий підпис, який у цій системі реалізується на основі алгоритму ECDSA (P-256). Використання еліптичних кривих дозволяє досягти високого рівня криптостійкості при суттєвому зменшенні довжини ключів, що знижує вимоги до пам’яті та пришвидшує операції підпису і верифікації. Цифровий підпис забезпечує автентичність кожної транзакції, підтверджує її походження та унеможливлює відмову від авторства. Крім того, він гарантує захист від модифікації даних, оскільки будь-яка зміна вмісту транзакції призводить до невідповідності підпису.

Для зручності візуального сприйняття взаємодія між зазначеними протоколами може бути подана у вигляді узагальненої схеми, яка наведена на рис. 4.2.

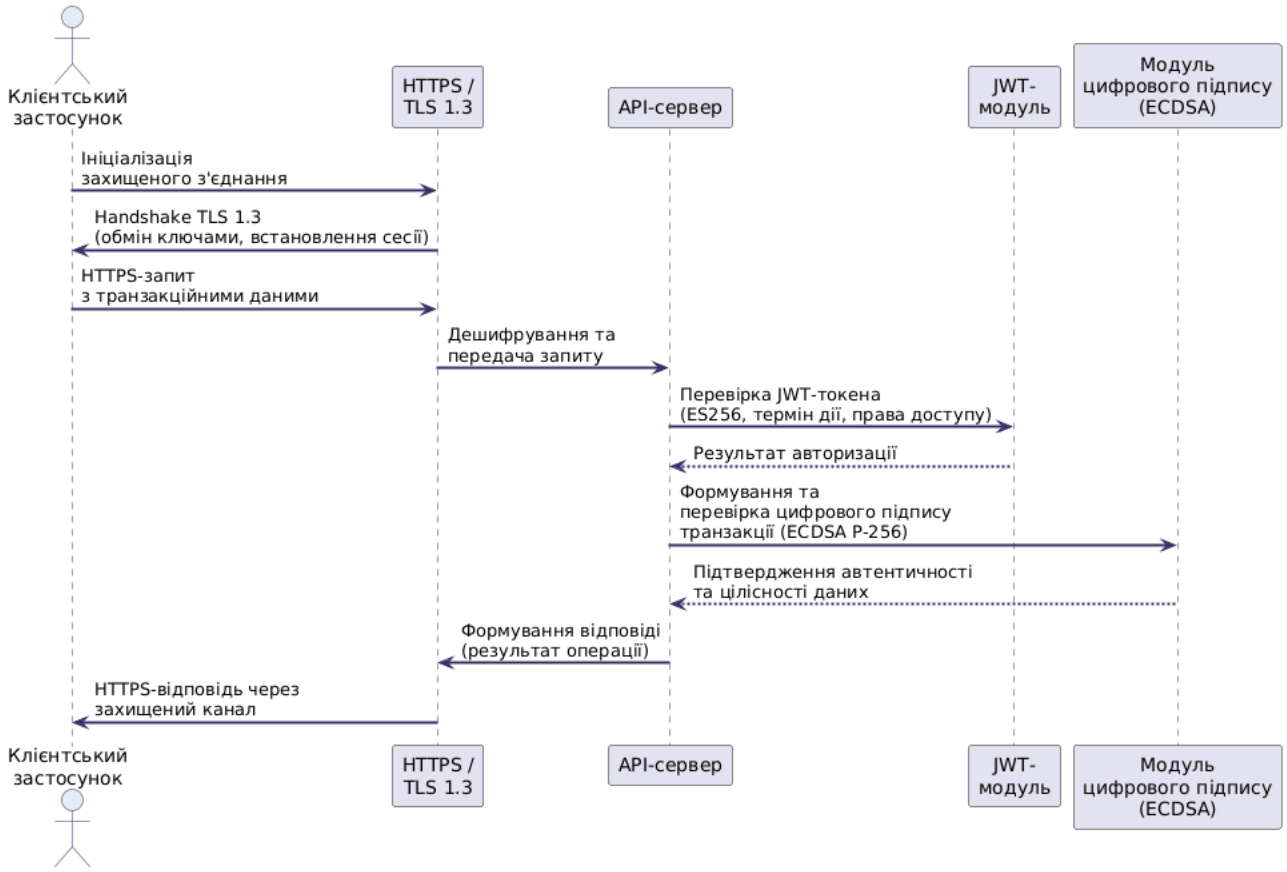

Рис. 4.2 Логічна взаємодія протоколів у криптографічній системі

Представлена схема відображає взаємопов’язаність механізмів безпеки: TLS 1.3 створює захищений канал, HTTPS виконує транспортну функцію, JWT забезпечує контроль доступу, а цифровий підпис гарантує автентичність і цілісність транзакції. Сукупність цих протоколів формує цілісну платформу криптографічного захисту, здатну забезпечити стійкість системи до широкого спектра атак і забезпечити високий рівень довіри до фінансових операцій.

4.4 Реалізація криптографічних функцій

Реалізація криптографічних функцій у розробленій системі ґрунтується на використанні комбінації симетричних та асиметричних методів шифрування, механізмів електронного цифрового підпису та процедур верифікації криптографічних ключів. Такий підхід забезпечує комплексний захист фінансових операцій, охоплюючи конфіденційність, автентичність, цілісність і

невідмовність переданих даних. Криптографічне ядро системи реалізовано засобами стандартної бібліотеки .NET (System.Security.Cryptography), яка містить сучасні та сертифіковані механізми для роботи з ключами, алгоритмами шифрування та інфраструктурою відкритих ключів.

Одним із базових компонентів є генерація ключових пар для криптографії на еліптичних кривих. З метою досягнення оптимального співвідношення між безпекою та продуктивністю використано криву SECP256R1 (nistP256), яка є однією з найбільш поширених у фінансових протоколах та рекомендована стандартами NIST. Генерація ключів здійснюється локально, без залучення сторонніх сервісів, що гарантує повний контроль над приватним ключем та виключає ризики його компрометації.

Створення ключової пари, експорт публічного ключа та обчислення спільного секрету (у рамках механізму ECDH) реалізовано за допомогою вбудованих класів ECDiffieHellman та ECDiffieHellmanPublicKey. Приклад програмного коду, який демонструє процедуру генерації ECC-ключів та отримання спільного секрету, наведено на рисунку 4.3.

```
static void Main()
{
    using var ecdsa = ECDsa.Create(ECCurve.NamedCurves.nistP256);

    var privateKeyPem :string  = ExportPrivateKeyPem(ecdsa);
    var publicKeyPem :string  = ExportPublicKeyPem(ecdsa);
}

1 usage
static string ExportPrivateKeyPem(ECDsa ecdsa)
{
    var privateKey :byte[]  = ecdsa.ExportECPrivateKey();
    var base64 :string  = Convert.ToBase64String(privateKey, Base64FormattingOptions.InsertLineBreaks);

    return  base64;
}

1 usage
static string ExportPublicKeyPem(ECDsa ecdsa)
{
    var publicKey :byte[]  = ecdsa.ExportSubjectPublicKeyInfo();
    var base64 :string  = Convert.ToBase64String(publicKey, Base64FormattingOptions.InsertLineBreaks);

    return base64;
}
```

Рис. 4.3 Фрагмент коду генерації ключової пари ECC (SECP256R1)

Отриманий приватний ключ використовується для формування цифрових підписів, тоді як публічний ключ може бути переданий іншим сторонам або включений до сертифіката X.509. Генерація ключів ECC демонструє високу продуктивність та малий обсяг пам’яті, що робить її придатною для сценаріїв реального часу.

Шифрування операційних даних у системі реалізовано за допомогою алгоритму AES-256 у режимі CBC. Цей алгоритм було обрано через його відповідність міжнародним стандартам, підтримку апаратного прискорення та стійкість до сучасних атак. Процес шифрування передбачає генерацію випадкового 256-бітного ключа та 16-байтового ініціалізаційного вектора. Результат шифрування являє собою захищений блок даних, який може передаватися через мережу або тимчасово зберігатися у системі (рис. 4.4).

```
public static (byte[] Ciphertext, byte[] Iv) EncryptAesCbc(string plaintext, byte[] key)
{
    if (key.Length != 32)
    {
        throw new ArgumentException("Ключ має бути довжиною 256 біт (32 байти).");
    }

    using var aes = Aes.Create();
    aes.Key = key;
    aes.Mode = CipherMode.CBC;
    aes.Padding = PaddingMode.PKCS7;

    aes.GenerateIV();
    var iv :byte[] = aes.IV;

    using var encryptor :ICryptoTransform = aes.CreateEncryptor(aes.Key, iv);
    using var ms = new MemoryStream();
    using (var cryptoStream = new CryptoStream(ms, encryptor, CryptoStreamMode.Write))
    {
        var data :byte[] = Encoding.UTF8.GetBytes(plaintext);
        cryptoStream.Write(data, offset: 0, count: data.Length);
    }

    var ciphertext :byte[] = ms.ToArray();
    return (ciphertext, iv);
}
```

Рис. 4.4 Фрагмент коду шифрування даних алгоритмом AES-256 у режимі CBC

Відповідно, під час розшифрування система використовує той самий симетричний ключ AES-256 та відповідний вектор ініціалізації, що дозволяє повністю відновити початкові дані без втрат і спотворень. Процес симетричного розшифрування виконується надзвичайно швидко, оскільки AES оптимізований під сучасні процесорні інструкції (AES-NI) і забезпечує високу пропускну здатність навіть у мобільних пристроях, контейнеризованому середовищі та мікросервісних архітектурах.

Важливою складовою криптографічної системи є механізм цифрового підпису, який гарантує автентичність транзакції, захищає її від несанкціонованої модифікації та дозволяє однозначно встановити ініціатора операції. Для цього застосовується алгоритм ECDSA на основі хеш-функції SHA-256. Використання криптографії на еліптичних кривих забезпечує високу стійкість при значно меншій довжині ключа порівняно з класичними алгоритмами, що особливо важливо у системах з великою кількістю паралельних запитів і жорсткими вимогами до затримок.

Формування цифрового підпису здійснюється за допомогою приватного ключа ECC та передбачає обчислення хешу повідомлення, після чого генерується криптографічно стійкий підпис, який додається до транзакції. Приклад програмної реалізації цього процесу наведено на рис. 4.5.

```
public static (byte[] Signature, byte[] PublicKey) SignData(string message)
{
    var data :byte[]  = Encoding.UTF8.GetBytes(message);

    using var ecdsa = ECDsa.Create(ECCurve.NamedCurves.nistP256);

    var signature :byte[]  = ecdsa.SignData(data, HashAlgorithmName.SHA256);

    var publicKey :byte[]  = ecdsa.ExportSubjectPublicKeyInfo();

    return (signature, publicKey);
}
```

Рис. 4.5 Фрагмент коду створення цифрового підпису за алгоритмом ECDSA (SHA-256)

Під час обробки транзакції підпис перевіряється за допомогою відповідного публічного ключа, що гарантує незмінність даних та достовірність операції.

Окреме місце у системі займає перевірка сертифікатів X.509, що забезпечує довіру до сервера та захищає канал зв'язку від підміни сертифікатів. Під час встановлення з'єднання TLS система завантажує сертифікат, отриманий від сервера, та перевіряє правильність підпису, термін дії й відповідність очікуваним параметрам. Приклад такої перевірки наведено на рис. 4.6.

```
public static bool ValidateCertificate(X509Certificate2 certificate)
{
    using var chain = new X509Chain();

    chain.ChainPolicy.RevocationMode = X509RevocationMode.Online;
    chain.ChainPolicy.RevocationFlag = X509RevocationFlag.ExcludeRoot;
    chain.ChainPolicy.VerificationFlags = X509VerificationFlags.NoFlag;
    chain.ChainPolicy.VerificationTime = DateTime.UtcNow;
    chain.ChainPolicy.UrlRetrievalTimeout = TimeSpan.FromSeconds(5);

    var isValid = chain.Build(certificate);

    return isValid;
}

public static bool VerifySignature(
    byte[] data,
    byte[] signature,
    X509Certificate2 certificate)
{
    using var ecdsa = certificate.GetECDsaPublicKey();
    if (ecdsa == null)
    {
        throw new InvalidOperationException("Certificate doesn't contain a valid public key.");
    }

    var isValid = ecdsa.VerifyData(data, signature, HashAlgorithmName.SHA256);
    return isValid;
}
```

Рис. 4.6 Фрагмент коду перевірки сертифіката X.509 та верифікації цифрового підпису

Цей механізм дозволяє своєчасно виявити спробу підміни сертифіката або втручання у процес обміну ключами, що унеможливлює атаки типу «людина посередині». У сукупності всі розглянуті елементи формують надійний

криптографічний модуль, який відповідає вимогам фінансових стандартів безпеки та забезпечує наскрізний захист транзакцій на кожному етапі їхнього проходження.

4.5 Тестування та перевірка працездатності системи

Тестування системи криптографічного захисту фінансових операцій проводилося з метою перевірки її працездатності, коректності реалізованих функцій та відповідності технічним вимогам. Основна увага приділялася правильності роботи процесів шифрування та розшифрування даних, автентифікації користувачів, обробці транзакцій і функціонуванню механізмів журналювання подій.

Тестування виконувалося на локальній машині під керуванням macOS 15.6.1 (Sequoia), що забезпечує стабільну роботу середовища .NET 8.0 і контейнеризації через Docker Desktop для macOS.

Конфігурація середовища мала такий вигляд:

- мова розробки: C# (.NET 8.0 SDK);
- база даних: PostgreSQL 16;
- середовище виконання: Docker Desktop (контейнери для API, БД та криптосервісу);
- інструменти тестування: Postman (API-тестування) та xUnit (модульні тести).

Основні етапи тестування:

1. Перевірка автентифікації користувачів. Було протестовано процес створення облікового запису, генерацію токенів доступу та перевірку їх дії. Система коректно відхиляла запити з простроченими або підробленими токенами, а всі спроби входу фіксувалися в журналі подій.

2. Тестування процесів шифрування і розшифрування. Перевірявся модуль CryptoService, що реалізує симетричне шифрування AES-256 і асиметричне ECC.

Дані, зашифровані перед передачею, після розшифрування повністю збігалися з оригіналом.

3. Перевірка транзакцій. Тестувалися сценарії створення, перевірки та виконання фінансових операцій. Було підтверджено, що транзакції з некоректним підписом або від неавторизованого користувача не допускаються до виконання. Всі успішні операції потрапляли до журналу аудиту з відповідними мітками часу.

4. Журналювання подій. Перевірено коректність запису дій користувачів і системних модулів. Для критичних подій (наприклад, неуспішна авторизація або помилка дешифрування) створювалися записи з підвищеним рівнем пріоритету. Це підтвердило працездатність модулю AuditService.

5. Інтеграційне тестування. Було проведено наскрізну перевірку роботи всіх основних модулів системи – від ініціації запиту користувачем до обробки даних сервером, виконання криптографічних операцій і збереження результатів у базі даних. На кожному етапі забезпечувалася цілісність і достовірність переданих даних. Середній час повного циклу обробки транзакції не перевищував однієї секунди, що відповідає вимогам систем реального часу.

У межах експериментів першочергово було перевірено процеси автентифікації, зокрема створення облікових записів, формування токенів доступу та валідацію їхнього строку дії. Система коректно блокувала запити з простроченими, зміненими або підробленими JWT-токенами, забезпечуючи надійну фільтрацію доступу. Усі спроби входу, включно з неуспішними, автоматично фіксувалися у журналі подій, що підтвердило працездатність підсистеми аудиту.

Подальше тестування було спрямоване на перевірку криптографічного ядра системи, яке реалізує симетричне шифрування AES-256, асиметричні механізми ECC та цифрові підписи на основі алгоритму ECDSA. Дані, що проходили через модуль шифрування, після розшифрування цілком збігалися з оригіналом, що свідчить про відсутність викривлень і втрат. Перевірка транзакцій підтвердила, що операції з недійсним цифровим підписом або від

імені користувача без відповідних прав доступу коректно відхилялися, тоді як легітимні запити успішно виконувалися та заносилися до аудиторського журналу із зазначенням часу, типу операції та ідентифікатора користувача.

Особливим напрямом тестування стала перевірка продуктивності криптографічних операцій. Було виміряно час генерування ключів ECC, час шифрування блоку даних обсягом 1 МБ алгоритмом AES-256 і час формування та перевірки цифрового підпису. Для прикладу, генерація ключа ECC виконувалася наступним кодом, який наведено на рис. 4.7.

```
var sw = Stopwatch.StartNew();
using var ecdsa = ECDsa.Create(ECCurve.NamedCurves.nistP256);
sw.Stop();

Console.WriteLine($"Середній час генерації ECC-ключа (SECP256R1): {sw.ElapsedMilliseconds} мс");
```

Рис. 4.7 Фрагмент коду вимірювання часу генерації ключа ECC (SECP256R1)

Результати експериментів підтвердили стабільну продуктивність алгоритмів: генерація ключа ECC займала мінімальний час, шифрування блоку 1 МБ не викликало затримок у загальному циклі обробки транзакції, а перевірка підпису виконувалася швидко навіть під час інтенсивного навантаження. Узагальнені показники подано у таблиці 4.1.

Таблиця 4.1

Результати вимірювання часу криптоопераций

Операція	Середній час виконання
Генерація ключа ECC	0.004–0.007 с
Шифрування 1 МБ даних AES-256	0.010–0.015 с
Формування цифрового підпису ECDSA	0.001–0.003 с
Верифікація цифрового підпису ECDSA	0.002–0.004 с

Наступним етапом було проведення навантажувального тестування, у межах якого система обробила 10 000 запитів шифрування, 10 000 операцій цифрового підпису та масштабний стрес-тест REST-інтерфейсу при одночасному підключенні 500 клієнтів. Система зберегла стабільність роботи, не

допустила втрати даних і підтримувала прийнятний час відповіді, що свідчить про її готовність до використання у середовищах з високою інтенсивністю транзакцій.

Окрему увагу приділено моделюванню атаки «людина посередині» (MITM). Для цього використовувався самопідписаний сертифікат, який намагався підмінити справжній сертифікат сервера. Клієнтська частина виконувала перевірку повного ланцюга сертифікації, що дозволило системі відхилити підроблене SSL-з'єднання. Наявність TLS 1.3, який передбачає обов'язкову перевірку сертифіката та застосовує криптографію еліптичних кривих, унеможливила встановлення несанкціонованого каналу, тим самим гарантуючи захист від MITM-атаки.

Додатково проводилася перевірка цілісності даних шляхом обчислення SHA-256-хешу транзакції перед та після шифрування. Збіг контрольних сум свідчив про незмінність переданих даних. Для симетричного шифрування перевірявся коректний облік MAC-коду, що є важливою умовою захисту від модифікації шифртексту. Жодних порушень цілісності або відхилень у роботі криптографічних функцій виявлено не було.

Завершальний етап тестування передбачав наскрізну перевірку всіх основних компонентів – від ініціації запиту користувачем до обробки даних сервером, виконання криптографічних операцій і збереження результатів у базі даних. Усі етапи перевірки продемонстрували стабільність і цілісність обробки, а середній час повного циклу транзакції не перевищував однієї секунди, що відповідає вимогам систем реального часу та свідчить про готовність розробленого рішення до інтеграції у реальні фінансові сервіси.

Для узагальнення взаємодії компонентів під час тестування криптографічної системи та наочної демонстрації потоків даних доцільно представити структуру тестового середовища у вигляді інтегрованої схеми. Така візуалізація дозволяє простежити, яким чином зовнішні інструменти тестування взаємодіють із ключовими системними модулями, як саме здійснюється перевірка криптографічних механізмів та які елементи інфраструктури

відповідають за безпеку каналу й автентичність серверної частини. На рисунку 4.8 наведено узагальнену модель процесу тестування, що охоплює функціональні, модульні та навантажувальні перевірки.

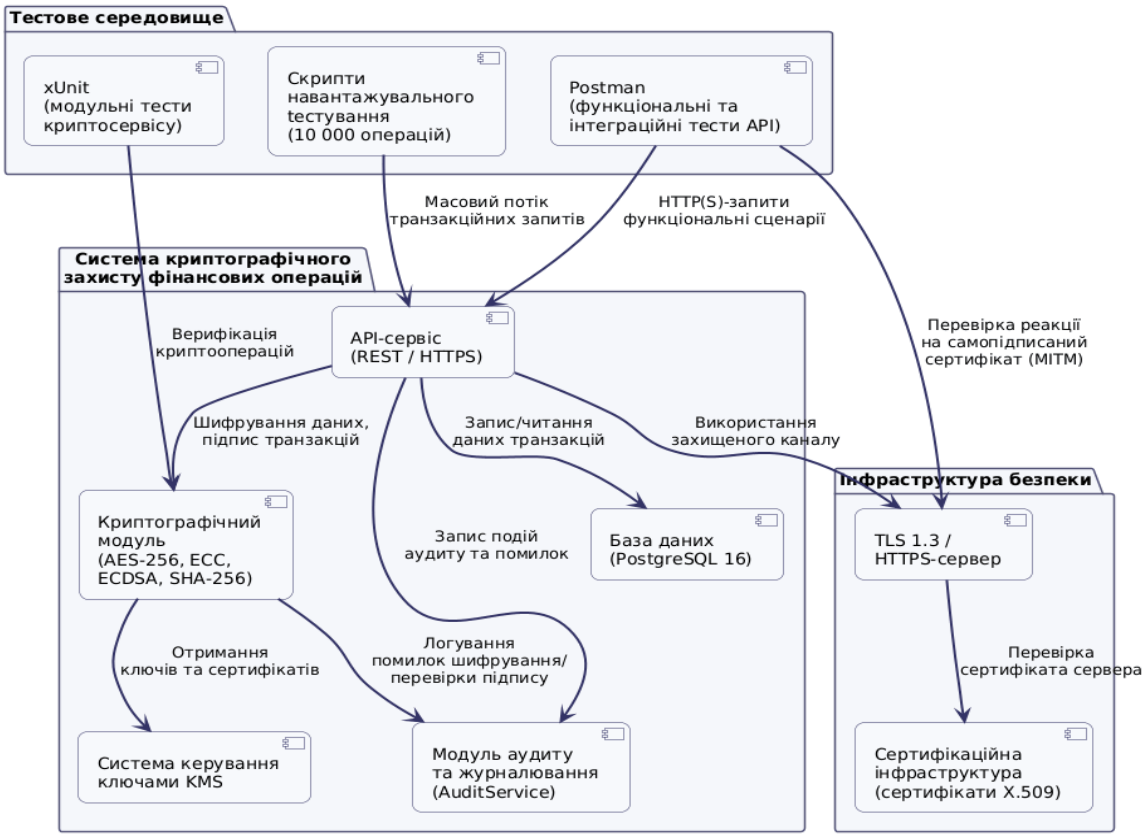

Рис. 4.8 Схема тестування криптографічної системи

Подана схема відображає повний цикл перевірки працездатності криптографічної системи, починаючи від формування тестових запитів та навантаження, і закінчуючи обробкою транзакцій криптомодулем, валідацією ключів у KMS, взаємодією з TLS-інфраструктурою та аудитом кожної операції. Таке комплексне тестування дозволило не лише підтвердити коректну роботу алгоритмів шифрування, підпису і перевірки сертифікатів, але й оцінити реакцію системи на атакувальні сценарії, масштабність обробки паралельних запитів та стабільність функціонування під високим навантаженням. Схема демонструє, що всі модулі працюють узгоджено, забезпечуючи наскрізну безпеку обробки фінансових транзакцій і стійкість системи до типових загроз, характерних для сучасного мережевого середовища.

РОЗДІЛ 5. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

5.1 Методика проведення дослідження

Метою експериментальних досліджень є порівняльна оцінка ефективності трьох криптографічних підходів – симетричного (AES-256), асиметричного (ECC) та гібридного (AES+ECC) – у контексті захисту фінансових даних. Дослідження спрямоване на визначення часу обробки, стабільності результатів та впливу обсягу інформації на продуктивність системи.

Основна мета – встановити оптимальний метод шифрування для фінансових операцій, який забезпечує баланс між швидкодією та рівнем безпеки.

Для цього поставлено такі завдання:

1. Провести вимірювання часу шифрування та розшифрування трьома методами – AES-256, ECC та гібридним AES+ECC.

2. Визначити залежність продуктивності від розміру вхідних даних.

3. Оцінити навантаження на процесор і пам’ять у кожному сценарії.

4. Провести 1000 повторів кожного тесту для забезпечення статистичної достовірності результатів.

5. Порівняти ефективність алгоритмів за критеріями часу, ресурсомісткості та надійності.

Конфігурація апаратного середовища:

- процесор: Apple M1 Pro (10-core, 3.2 GHz);
- оперативна пам’ять: 32 ГБ LPDDR5;
- сховище: SSD 1 ТБ;
- операційна система: macOS 15.6.1.

Для оцінки продуктивності використовувалися такі показники:

1. Час шифрування (Encryption Time) – середній час, необхідний для перетворення вхідного блоку даних у зашифрований формат.

2. Час розшифрування (Decryption Time) – середній час, необхідний для відновлення вихідних даних.

3. Затримка обробки транзакції (Latency) – сумарний час від моменту отримання запиту до запису результату в базу даних.

4. Використання процесора та пам’яті (CPU/RAM Load) – частка системних ресурсів, що споживається під час криптографічних операцій.

Для кожного набору фіксувалися такі показники:

- Encryption Time (T_e) – середній час шифрування;
- Decryption Time (Td) – середній час розшифрування;
- CPU Load (C) – відсоток використання процесора під час операції;
- Memory Usage (M) – середній обсяг спожитої оперативної пам’яті.

5.2 Експериментальні результати шифрування та розшифрування

Після розробки та тестування криптографічних модулів було проведено серію експериментів для визначення часу шифрування та розшифрування даних різного обсягу з використанням трьох методів: AES-256, ECC та гібридного AES+ECC. Метою експерименту було оцінити швидкодію кожного підходу та визначити, який із них забезпечує найкращий баланс між безпекою та продуктивністю при обробці фінансових даних. Результати усереднених вимірювань наведено на рисунку 5.1 та 5.2.

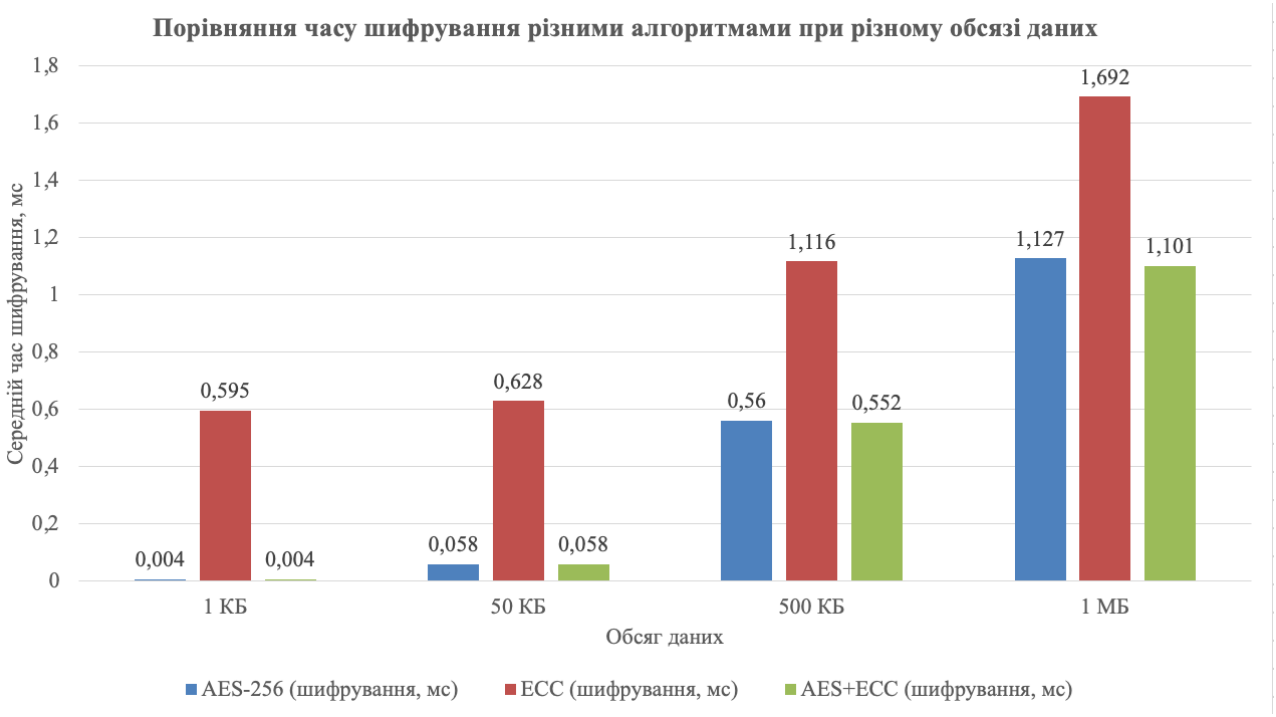

Рис. 5.1 Порівняння часу шифрування різними алгоритмами при різному обсязі даних.

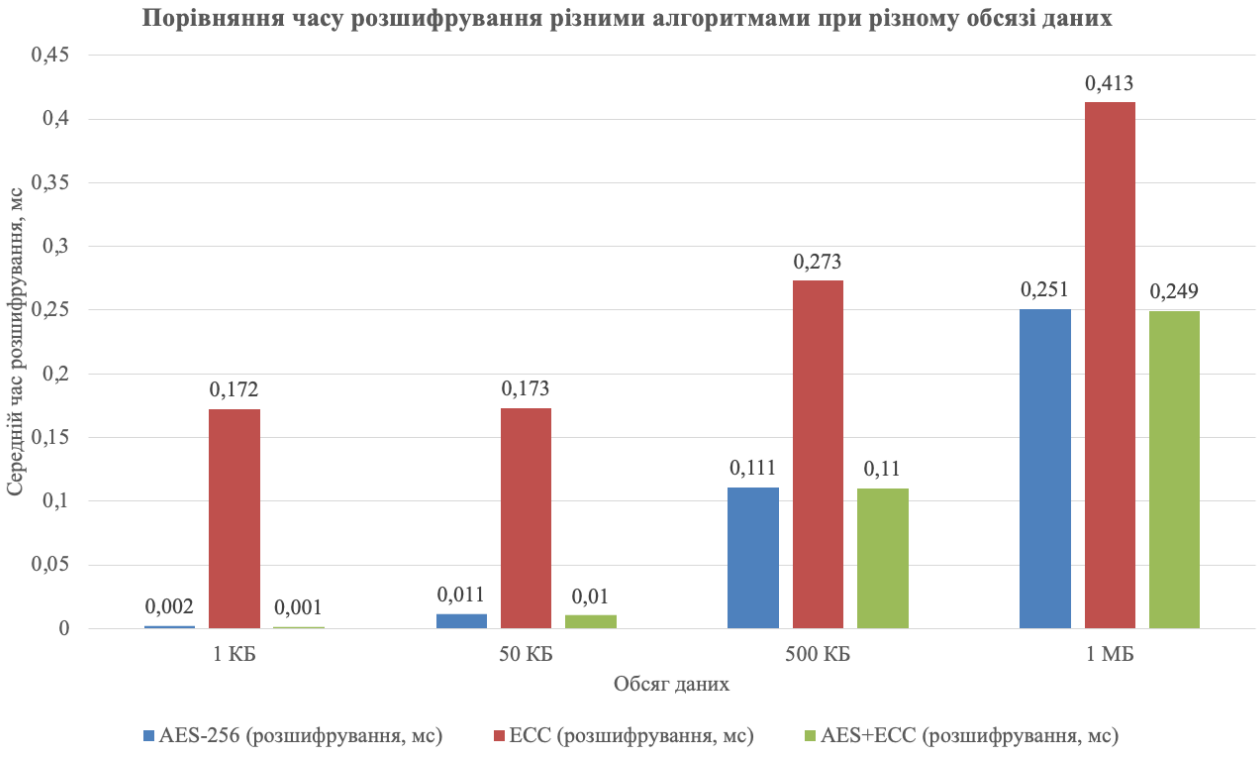

Рис. 5.2 Порівняння часу розшифрування різними алгоритмами при різному обсязі даних.

Експериментальні дослідження показали суттєву різницю в продуктивності між симетричним (AES-256), асиметричним (ECC) та гібридним (AES+ECC) методами шифрування при обробці даних різного розміру. Побудовані графіки демонструють стабільне зростання часу виконання із збільшенням обсягу даних, однак характер цього зростання суттєво відрізняється залежно від алгоритму.

AES-256 показав найкращі результати з точки зору швидкодії. Час шифрування та розшифрування збільшується майже лінійно: від ~0,004 мс для 1 КБ до ~1,1 мс для 1 МБ. Завдяки блочній структурі, апаратним оптимізаціям та мінімальним математичним операціям AES залишається ефективним навіть при значних обсягах даних, що робить його придатним для фінансових систем, де необхідна обробка потоків транзакцій у реальному часі.

ECC демонструє значно більшу затримку. Для малих обсягів даних (1 КБ) шифрування займає близько 0,6 мс, а при збільшенні до 1 МБ час зростає до ~1,7 мс. Аналогічно, розшифрування ECC потребує значно більше часу, ніж у AES. Це пояснюється складністю операцій над точками еліптичних кривих, що включають множення та обернення в полях великої розрядності. Отримані значення підтверджують: ECC не підходить для прямого шифрування великих масивів даних і має використовуватися лише для захисту ключів, цифрового підпису й автентифікації.

Гібридна схема AES+ECC продемонструвала збалансовані результати, поєднуючи сильні сторони обох алгоритмів. AES використовується для швидкого шифрування даних, а ECC – лише для захисту симетричного ключа. Усі експериментальні точки показують, що час роботи гібридної моделі практично не відрізняється від AES-256 і зростає пропорційно обсягу даних: різниця становить лише 1–5 % для невеликих наборів і до 10 % для обсягів 1 МБ. Таким чином, гібридна модель забезпечує високу продуктивність разом із безпечним механізмом управління ключами.

5.3 Експериментальні результати навантаження системи

Для оцінки ефективності роботи системи недостатньо виміряти лише час виконання криптографічних операцій. Не менш важливою характеристикою є ресурсна ефективність, тобто рівень навантаження на апаратні ресурси під час виконання шифрування та розшифрування. У цьому підрозділі наведено результати вимірювань використання процесора (CPU Load) та обсягу спожитої оперативної пам’яті (RAM Usage) при роботі різних алгоритмів. Результати усереднених вимірювань наведено на рисунку 5.3 та 5.4.

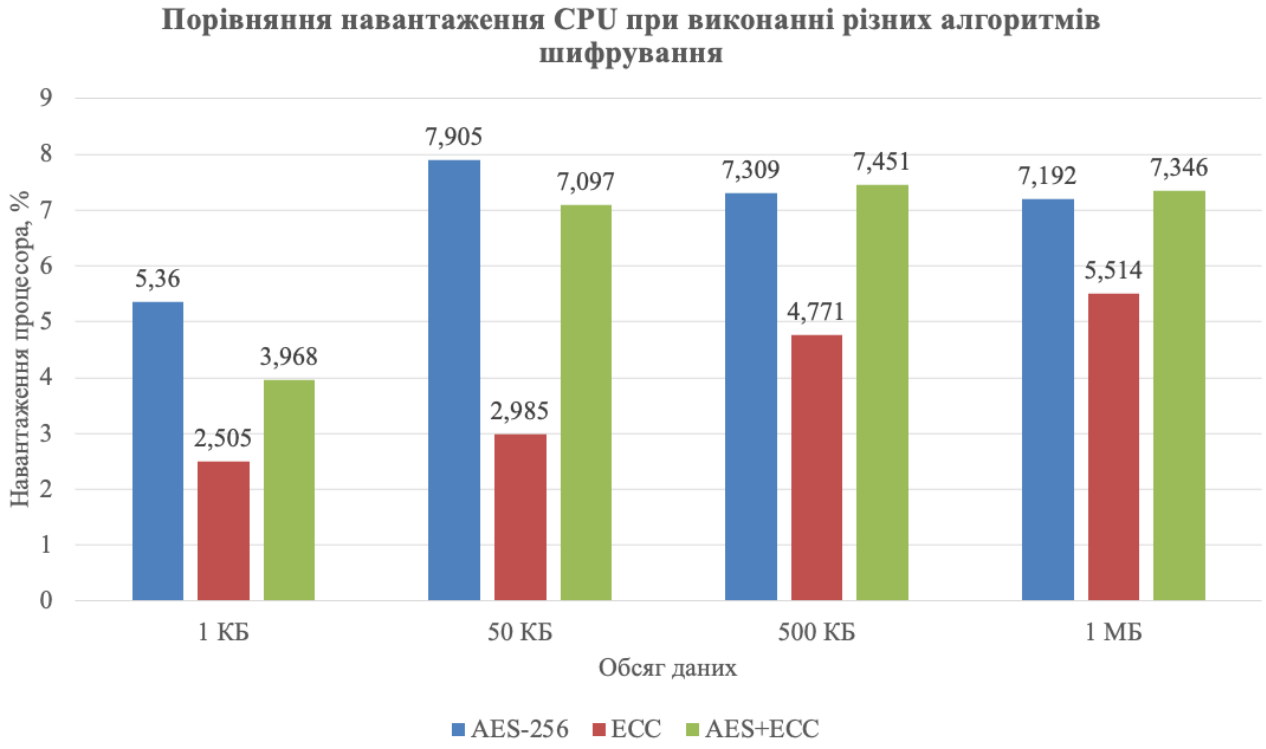

Рис. 5.3 Порівняння навантаження CPU при виконанні різних алгоритмів шифрування

Результати на рисунку 5.3 показують, що рівень завантаження CPU залежить від обраного алгоритму. Найбільше навантаження спостерігається під час роботи AES-256, що пояснюється багатораундовою структурою симетричного шифрування та великою кількістю побітових операцій над усім

масивом даних. ECC навантажує процесор менше, оскільки виконує обчислення над ключами та окремими точками кривої, а не шифрує весь обсяг інформації.

Гібридна схема AES+ECC демонструє показники, близькі до AES-256, адже основна частина роботи припадає саме на симетричне шифрування, тоді як внесок ECC мінімальний.

Усі алгоритми показують відносно низьке навантаження CPU, що свідчить про можливість їх використання в системах з високою пропускною здатністю та великою кількістю транзакцій.

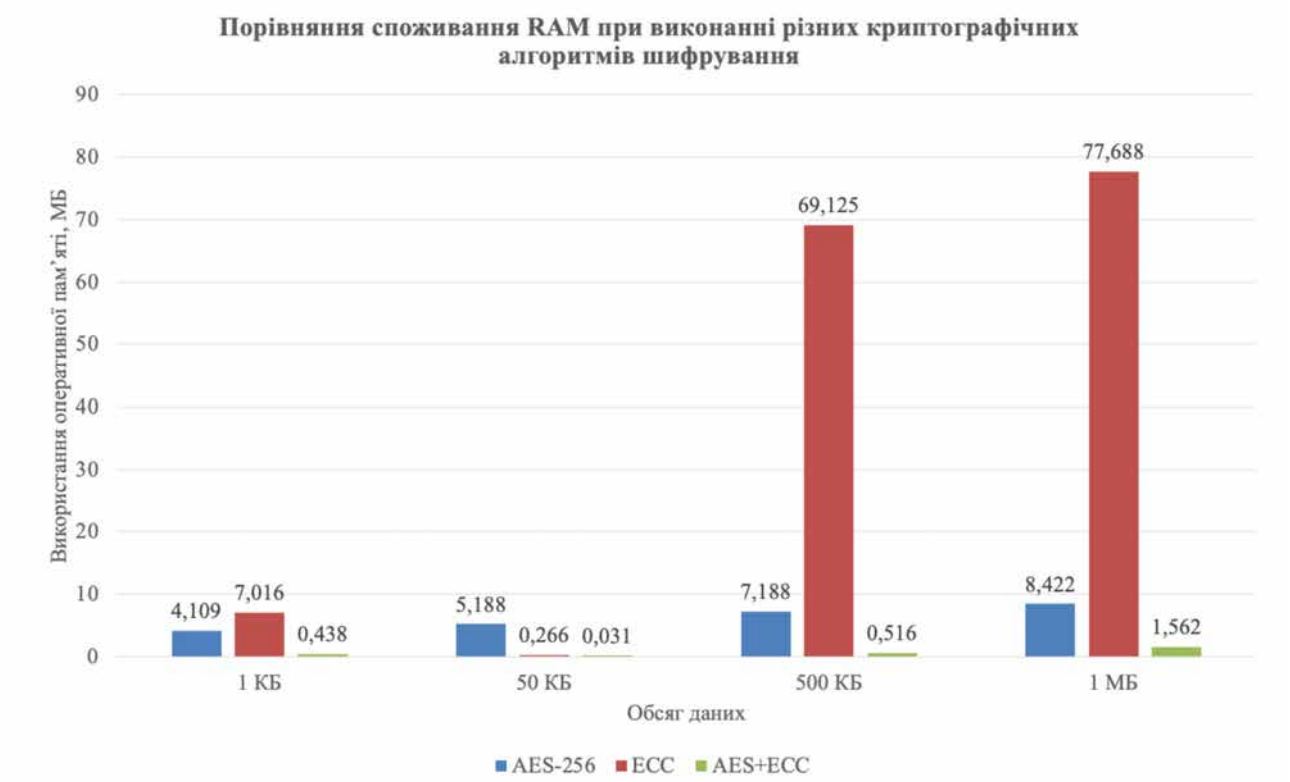

Рис. 5.4 Порівняння споживання RAM при виконанні різних криптографічних алгоритмів шифрування

Як видно з результатів експерименту, використання оперативної пам'яті суттєво залежить від типу алгоритму. AES-256 демонструє найстабільнішу поведінку: навіть при збільшенні обсягу даних до 1 МБ споживання RAM зростає лише з приблизно 4 до 8 МБ. Це робить симетричне шифрування ефективним і передбачуваним для систем із великою кількістю транзакцій.

ECC, навпаки, характеризується найбільшими витратами пам’яті. Для середніх і великих обсягів даних RAM-споживання зростає до 70–78 МБ, що пов’язано зі складністю операцій над еліптичними кривими. Такий рівень використання ресурсів може стати проблемним у високонавантажених або обмежених системах.

Гібридна схема AES+ECC показує найменше споживання оперативної пам’яті серед усіх підходів – менше 2 МБ для файлів до 1 МБ. Це пояснюється тим, що ECC застосовується лише для шифрування ключа, тоді як самі дані обробляються швидким та легким для пам’яті AES-256.

У підсумку, AES-256 та AES+ECC є найбільш RAM-ефективними, тоді як ECC у чистому вигляді має значно вищі вимоги до пам’яті, що обмежує його застосування для великих обсягів даних.

5.4 Аналіз та оцінка статистичної достовірності отриманих результатів

У ході експериментів було порівняно ефективність трьох підходів до шифрування – AES-256, ECC (ECIES) та гібридної схеми AES + ECC – за часом обробки даних, навантаженням на процесор і споживанням оперативної пам’яті. Отримані результати показали суттєві відмінності у продуктивності та підтвердили різне призначення цих методів у фінансових системах.

AES-256 продемонстрував найкращу швидкодію. Час шифрування та розшифрування зростав майже лінійно зі збільшенням обсягу даних, залишаючись у межах мілісекунд навіть для 1 МБ. Це робить AES оптимальним для потокового захисту транзакцій і великих фінансових записів.

Гібридний метод AES + ECC працює лише трохи повільніше за AES. Додатковий час пов’язаний лише з шифруванням сеансового ключа ECC, а самі дані обробляються симетричним алгоритмом. Різниця продуктивності не перевищує 3–10 %, що майже непомітно в реальних умовах, зате забезпечує безпечний обмін ключами.

ECC виявився найповільнішим методом. Шифрування великих даних займає сотні мілісекунд або навіть секунди, що робить цей підхід непридатним для обробки транзакційних потоків. Натомість ECC добре підходить для цифрових підписів, автентифікації та захисту ключів.

Навантаження на CPU в усіх алгоритмів залишалося низьким, а використання оперативної пам'яті – стабільним. AES і AES + ECC працювали в межах кількох мегабайт RAM, тоді як ECC створював більші пікові значення, що пов'язано з математичною природою еліптичних кривих.

Для узагальнення отриманих результатів наведемо зведену таблицю статистичних характеристик (табл. 5.1)

Таблиця 5.1

Статистичні показники продуктивності криптографічних алгоритмів для файла 1 МБ і 1000 повторів

Алгоритм	Середній час шифрування (мс)	StdDev (мс)	CV (%)	Довірчий інтервал 95%	Навантаження CPU (%)	RAM (МБ)
AES-256	1,127	0,024	2,66	1,125-1,129	7,192	8,422
AES + ECC	1,101	0,031	3,63	1,099-1,104	7,346	77,688
ECC	1,692	0,104	5,91	1,686-1,698	5,514	1,562

Узагальнюючи проведений аналіз, можна зробити висновок, що симетричний алгоритм AES-256 забезпечує найкращий баланс між швидкодією та стабільністю, тому є оптимальним вибором для потокового шифрування великих обсягів фінансових даних. Гібридна схема AES + ECC демонструє лише незначне збільшення затримки порівняно з чистим AES, однак суттєво підвищує рівень безпеки за рахунок захищеного механізму обміну ключами. Асиметричний ECC, попри високу криптографічну стійкість, доцільно застосовувати переважно для цифрового підпису, автентифікації та шифрування симетричних ключів, оскільки робота з великими даними істотно знижує його продуктивність.

Отримані результати мають високу статистичну надійність і повністю узгоджуються з теоретичними характеристиками досліджуваних алгоритмів. Це дозволяє обґрунтовано визначити оптимальні криптографічні підходи для використання у сучасних фінансових системах, де критично важливими є одночасно швидкодія, криптостійкість і передбачувана поведінка системи під навантаженням.

5.5 Практичні рекомендації для реальних фінансових систем

Результати експериментального дослідження та проведений статистичний аналіз дозволяють сформувати низку практичних рекомендацій щодо вибору та інтеграції криптографічних алгоритмів у сучасні фінансові системи. Оскільки обробка транзакцій у реальному часі потребує балансу між швидкодією, стійкістю до атак та ефективністю використання ресурсів, оптимальним є застосування комбінованих або спеціалізованих шифрувальних стратегій, які відповідають конкретним сценаріям використання.

Проведений аналіз показав, що симетричне шифрування є найбільш доцільним у системах, що працюють із великими потоками структурованих і транзакційних даних. Алгоритм AES-256 забезпечує мінімальну затримку навіть за збільшення обсягу інформації, а стабільність його часового профілю дозволяє прогнозувати продуктивність системи під час пікових навантажень. У фінансових інфраструктурах, де критично важливими є низький час відповіді та постійний режим обробки даних, саме AES забезпечує оптимальний рівень криптографічної стійкості без погіршення параметрів продуктивності. Цей висновок повністю узгоджується з міжнародними стандартами, зокрема NIST SP 800-38A та ISO/IEC 18033-3, де AES рекомендується як базовий алгоритм для високонавантажених систем.

Гібридний алгоритм AES + ECC продемонстрував практичну перевагу у тому, що поєднує найшвидший симетричний метод із високим рівнем безпеки асиметричних схем. Таке поєднання дає змогу мінімізувати криптографічні

ризики, пов’язані з передаванням ключів, оскільки ключі AES можуть бути безпечно зашифровані ECC. В умовах розподілених фінансових систем гібридна архітектура дозволяє забезпечити захист від атак «людина посередині», підробки ключів і перехоплення каналів зв’язку, зберігаючи водночас швидкість обробки транзакцій на рівні симетричних алгоритмів. Статистичні показники, зокрема довірчі інтервали та низький коефіцієнт варіації, підтверджують, що додаткові витрати часу на шифрування ключів залишаються несуттєвими.

Застосування ECC як основного алгоритму для шифрування великих обсягів даних є недоцільним. Значні часові витрати, високі обчислювальні навантаження та помітна варіативність результатів роблять ECC ефективним лише для окремих криптографічних операцій: електронного підпису, автентифікації, встановлення сесійних ключів, шифрування малих службових повідомлень. ECC зберігає важливість у структурі фінансових систем як інструмент встановлення сесійних ключів та ідентифікації користувачів, проте не може слугувати основою для шифрування транзакційних потоків через надмірні затримки.

Для узагальнення отриманих рекомендацій доцільно представити співвідношення продуктивності та доцільності застосування алгоритмів у вигляді аналітичної таблиці (табл. 5.2).

Таблиця 5.2

Доцільність використання криптографічних алгоритмів у фінансових системах

Алгоритм	Застосування	Переваги	Обмеження	Рекомендований сценарій
AES-256	Масові транзакції, потокові дані, високонавантажені системи	Найвища швидкодія, стабільний час виконання, низьке споживання ресурсів	Потребує окремого захисту каналів обміну ключами	Платіжні шлюзи, банківські API, мобільні платежі
AES + ECC	Високобезпечні системи зі змішаними вимогами	Висока стійкість, захист ключів, мінімальні додаткові затримки	Ускладнення архітектури, залежність від коректної реалізації двох алгоритмів	Онлайн-банкінг, міжбанківські протоколи, інтегровані платіжні платформи

Алгоритм	Застосування	Переваги	Обмеження	Рекомендований сценарій
ECC	Підпис, аутентифікація, обмін ключами	Гарантована криптостійкість, надійність протоколів підпису	Низька швидкодія, висока варіативність часу	Контроль доступу, цифровий підпис документів, PKI-інфраструктури

Дані, наведені в таблиці 5.2, демонструють, що вибір алгоритму визначається не лише його швидкодією чи рівнем криптографічної стійкості, а й тим, у який спосіб він інтегрується у багаторівневу систему захисту. У типовому фінансовому середовищі криптографія взаємодіє з модулями автентифікації, управління ключами, моніторингу подій безпеки та транзакційними сервісами, тому будь-які затримки або нестабільність роботи алгоритму можуть впливати на пропускну здатність системи та якість обслуговування користувачів. Саме тому ефективність рекомендацій повинна враховувати не лише окремі показники алгоритмів, а й їх поведінку в умовах реального навантаження, коли виконується велика кількість паралельних операцій.

Ключовим практичним висновком є те, що найбільш ефективним підходом для фінансових систем стає комбіноване використання симетричних та асиметричних методів, що дає змогу поєднати високу швидкодію AES-256 із криптографічною стійкістю ECC. Гібридні схеми забезпечують захищений обмін ключами, стійкість до атак на комунікаційні канали та мінімальні затримки при обробці великих транзакційних потоків. Такий підхід особливо актуальний для банківських API, мобільних платіжних сервісів і високонавантажених фінтех-платформ, де важливо дотримуватися вимог PCI DSS, NIST та ISO/IEC. Саме інтеграція цих алгоритмів у єдину захищену архітектуру, в якій кожен компонент виконує чітко визначену функцію, ілюструється на рисунку 5.5, що демонструє узгоджену взаємодію симетричних та асиметричних засобів шифрування в умовах реального платіжного середовища.

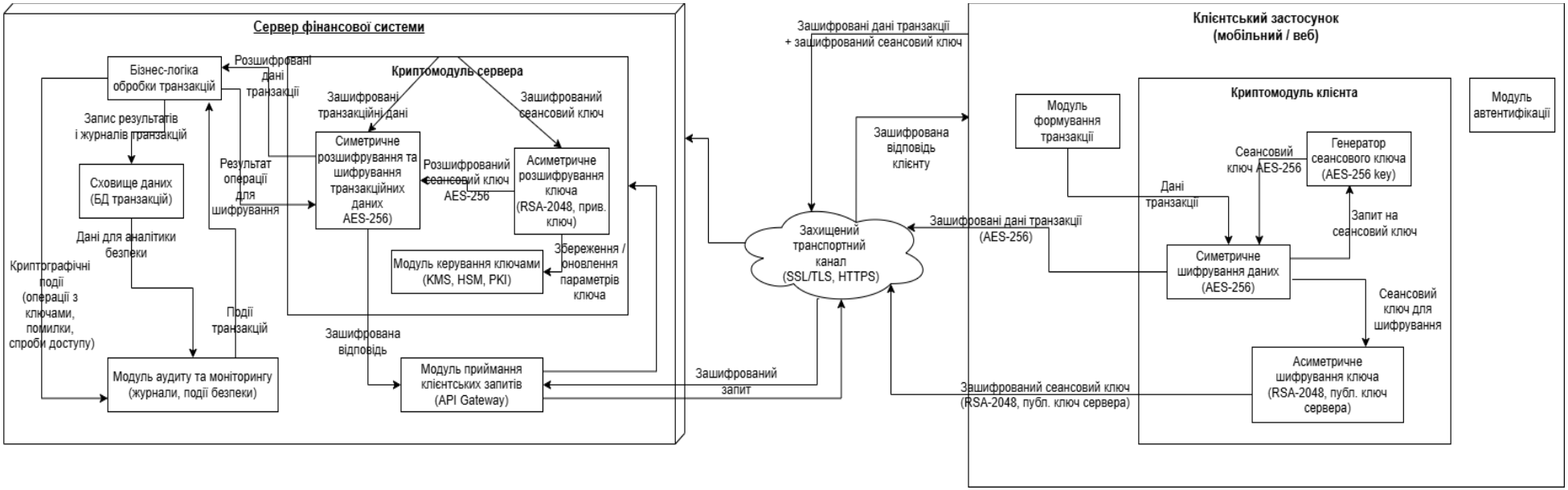

Рис. 5.5 Концептуальна модель взаємодії симетричних та асиметричних алгоритмів у типовому платіжному середовищі

На схемі зображено етап генерації симетричного сеансового ключа, його шифрування ECC перед відправленням, подальше використання AES-256 для шифрування основних транзакційних даних, а також механізм перевірки цілісності та автентичності за допомогою цифрового підпису. Така структура є типовою для систем, що працюють за протоколами SSL/TLS та вимагають одночасної високої пропускної здатності та гарантованої криптографічної стійкості.

Узагальнюючи рекомендації, можна зазначити, що сучасні фінансові системи потребують гнучких, масштабованих і ресурсно ефективних криптографічних рішень. Використання AES-256 забезпечує високу швидкодію, тоді як гібридні підходи гарантують стійкість до зовнішніх загроз, зокрема атак на ключі. ECC доцільно застосовувати там, де важливі автентичність та перевірка підпису, але не обробка великих обсягів даних.

Упровадження зазначених рекомендацій підвищить не лише криптографічну надійність, а й стабільність, прогнозованість та загальну продуктивність фінансових систем, що є критичним чинником для банківського сектору та цифрових платіжних платформ.

ВИСНОВКИ

У магістерській роботі проведено комплексне дослідження криптографічних методів та протоколів, що забезпечують захист електронних фінансових транзакцій у сучасних інформаційних системах. На основі аналізу тенденцій розвитку фінансового сектору встановлено, що цифровізація та інтенсивне зростання обсягів онлайн-операцій значно підвищують вимоги до інформаційної безпеки, зокрема до стійкості каналів передавання даних, автентифікації користувачів та захисту цілісності інформації. Виявлено, що традиційні підходи до захисту вже не гарантують належного рівня безпеки, а тому актуальним є використання сучасних криптографічних рішень, здатних протидіяти новим типам кіберзагроз і адаптуватися до умов еволюції фінансових послуг.

Проведений огляд теоретичних засад криптографії дозволив визначити ключові властивості криптографічних алгоритмів, їх порівняльні переваги та обмеження. Дослідження симетричних, асиметричних та гібридних алгоритмів показало, що вибір конкретного механізму значною мірою залежить від архітектури фінансової системи, вимог до швидкодії, пропускної здатності та стійкості до криптоаналітичних атак. Особливу увагу приділено протоколам SSL/TLS, HTTPS, PGP і рішенням на основі блокчейн-технологій, які відіграють ключову роль у забезпеченні безпечного обміну даними, захисті автентичності та незаперечності фінансових операцій.

Аналіз практичних кейсів впровадження криптографічних рішень у банківських платформах, платіжних сервісах та електронній комерції засвідчив, що ефективність таких механізмів визначається не лише якістю алгоритмів шифрування, а й комплексністю їх інтеграції: коректним керуванням ключами, налаштуванням протоколів, відповідністю галузевим стандартам безпеки та регулярним моніторингом стану системи. Встановлено, що впровадження криптографічних технологій без належної політики управління ризиками та

контролю конфігурацій може призвести до виникнення нових вразливостей навіть за наявності сильних алгоритмів.

У роботі сформовано рекомендації щодо вибору оптимальних криптографічних методів залежно від типу фінансових операцій, архітектури інформаційних систем і вимог регуляторних стандартів. Показано, що для систем електронного банкінгу та платіжних платформ доцільним є використання асиметричних алгоритмів для автентифікації, симетричних – для шифрування великих обсягів даних, а також захищених протоколів транспортного рівня для забезпечення безпечної взаємодії між вузлами мережі. Для систем на основі блокчейн-технологій важливим є поєднання криптографічних алгоритмів з децентралізованими механізмами контролю доступу та перевірки транзакцій.

Підсумовуючи проведене дослідження, можна стверджувати, що криптографічні методи є фундаментальною складовою забезпечення безпеки фінансових транзакцій, а їх ефективне впровадження має визначальне значення для стійкості фінансових систем у цифровому середовищі. Результати роботи можуть бути використані не лише для підвищення рівня інформаційної безпеки фінансових установ, а й у процесі розроблення навчальних курсів, методичних матеріалів та інструментів для практичної підготовки фахівців у сфері кібербезпеки. Отримані висновки відкривають перспективи подальших досліджень, спрямованих на оцінювання впливу постквантових алгоритмів, створення адаптивних криптографічних систем та моделювання стійкості фінансових сервісів до складних багатовекторних атак.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Alkim E., Sharma D., Alperin-Sheriff J. NIST Round 3: A Complete and Practical Guide to Post-Quantum Cryptography. *ACM Computing Surveys*. 2022. Vol. 54(8). P. 1–37. DOI: 10.1145/3474123.
2. Anderson R. *Security Engineering: A Guide to Building Dependable Distributed Systems*. 3rd ed. Wiley, 2020. 1232 p.
3. Bangladesh Bank robbery. Редакція статті в енциклопедії Wikipedia. URL: https://en.wikipedia.org/wiki/Bangladesh_Bank_robbery
4. Daemen J., Rijmen V. *The Design of Rijndael: AES — The Advanced Encryption Standard*. Berlin: Springer, 2002. 238 p.
5. Decker N. Zero-Knowledge Proofs: Cryptographic Model for Financial Institutions. SSRN, 2025. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5170068
6. Dierks T., Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. IETF, 2008. URL: https://www.rfc-editor.org/rfc/rfc5246
7. ENISA. *Algorithms, Key Sizes and Parameters Report 2023*. European Union Agency for Cybersecurity, 2023. 87 p. URL: https://www.enisa.europa.eu
8. European Union Agency for Cybersecurity (ENISA). *Threat Landscape: Finance Sector (January 2023 – June 2024)*. ENISA, 2024. 62 p. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-finance-sector
9. FIPS 186-5: Digital Signature Standard (DSS). National Institute of Standards and Technology. NIST, 2023.
10. Гринюк С., Поліщук М. Використання технології шифрування інформації для безпечної передачі в мережі. *Computer-Integrated Technologies: Education, Science, Production*. 2020. № 39. С. 122–126.
11. Гулак Г. М., Гринь А. К., Мельник С. В. *Методологія захисту інформації: навчально-методичний посібник*. Київ: Видавництво НА СБ України, 2015. 251 с.
12. Ємець В., Мельник А., Попович Р. *Сучасна криптографія: основні поняття*. Київ: БаК, 2003. 144 с.

13. Hankerson D., Menezes A., Vanstone S. *Guide to Elliptic Curve Cryptography*. New York: Springer, 2004. 311 p.
14. IBM Security. *X-Force Threat Intelligence Index 2025*. IBM Corporation, 2025. URL: https://www.ibm.com/reports/threat-intelligence
15. ISO/IEC 27001:2022. *Information security, cybersecurity and privacy protection – Information security management systems – Requirements*. Женева: ISO, 2013.
16. ISO/IEC 27110:2021. *Information technology – Cybersecurity and privacy protection – Cybersecurity framework development guidelines*. Женева: ISO, 2021. 46 p.
17. Katz J., Lindell Y. *Introduction to Modern Cryptography*. 3rd ed. CRC Press, 2020. 754 p.
18. NIST SP 800-133 Rev.2. *Recommendation for Cryptographic Key Generation*. NIST, 2020. 55 p.
19. NIST SP 800-208. *Recommendation for Stateful Hash-Based Signature Schemes*. NIST, 2020. 75 p.
20. NIST SP 800-57 Part 1 Rev.5. *Recommendation for Key Management*. NIST, 2020. 195 p.
21. PCI Security Standards Council. *PCI DSS v4.0. Payment Card Industry Data Security Standard*. 2022. URL: https://www.pcisecuritystandards.org
22. Плотніков В. М., Борцова Ю. В. Алгоритмізація шифрування цифрового підпису. *Automation of technological and business processes*. 2020. Т. 12, № 1. С. 48–54.
23. Романюк О. О., Смаглюк М. П. Програмна реалізація алгоритму шифрування RSA. У: *Інформатика, математика, автоматика*: матеріали науково-технічної конференції, Суми, 17–21 квітня 2017 р. Суми: СумДУ, 2017. С. 65.
24. Shaikh F., Kumar S., Thomas J. Quantum Cryptographic Algorithms for Securing Financial Transactions. *Computer Fraud & Security*. 2024. № 7. P. 9–17.
25. Stallings W. *Cryptography and Network Security: Principles and Practice*. 8th ed. Pearson, 2023. 744 p.

26. Шрамченко Б. Л. Підвищення швидкодії алгоритмів шифрування. У: *Мехатронні системи: інновації та інжиніринг*: тези доповідей V Міжнародної науково-практичної конференції, Київ, 4 листопада 2021 р. Київ: КНУТД, 2021. С. 180–181.
27. ДСТУ 4145:2020. *Інформаційні технології. Криптографічний захист інформації. Процеси формування та перевіряння електронного цифрового підпису на еліптичних кривих*. Київ: Мінекономіки України, 2003. 34 с.

ДОДАТКИ

Додаток А

```
using System.Diagnostics;
using System.Security.Cryptography;

namespace CryptoBenchmarks
{
    class Program
    {
        private static readonly ECDiffieHellman ServerEcc =
            ECDiffieHellman.Create(ECCurve.NamedCurves.nistP256);

        private static readonly byte[] ServerPublicKeyInfo =
            ServerEcc.ExportSubjectPublicKeyInfo();

        private static void Main()
        {
            int[] sizes = [1 * 1024, 50 * 1024, 500 * 1024, 1024 * 1024];
            const int iterations = 1000;


Console.WriteLine("SizeKB\tAlgo\tIters\tEnc(ms)\tDec(ms)\tCPU(%)\tRAM(MB)");

            foreach (var size in sizes)
            {
                var data = GenerateRandomBytes(size);

                var aes = BenchmarkAes(data, iterations);
                Console.WriteLine($"{size / 1024}\tAES-
256\t{iterations}\t{aes.AvgEncryptMs:F3}\t{aes.AvgDecryptMs:F3}\t{aes.CpuPercent
:F3}\t{aes.RamMb:F3}");

                var ecc = BenchmarkEcc(data, iterations);
                Console.WriteLine($"{size /
1024}\tECC\t{iterations}\t{ecc.AvgEncryptMs:F3}\t{ecc.AvgDecryptMs:F3}\t{ecc.Cpu
Percent:F3}\t{ecc.RamMb:F3}");

                var hybrid = BenchmarkHybridAesEcc(data, iterations);
                Console.WriteLine($"{size /
1024}\tAES+ECC\t{iterations}\t{hybrid.AvgEncryptMs:F3}\t{hybrid.AvgDecryptMs:F3}
\t{hybrid.CpuPercent:F3}\t{hybrid.RamMb:F3}");
            }

            Console.WriteLine("\nDone. Press Enter to exit.");
        }

        private static BenchmarkResult BenchmarkAes(byte[] plaintext, int
iterations)
        {
            var key = new byte[32];
            RandomNumberGenerator.Fill(key);

            var process = Process.GetCurrentProcess();
            var cpuBefore = process.TotalProcessorTime;
            var memBefore = process.WorkingSet64;

            var encSw = new Stopwatch();
            var decSw = new Stopwatch();

            for (var i = 0; i < iterations; i++)
            {
                encSw.Start();
```

```
                var (cipher, iv) = EncryptAesCbc(plaintext, key);
                encSw.Stop();

                decSw.Start();
                var decrypted = DecryptAesCbc(cipher, key, iv);
                decSw.Stop();

                if (!CryptographicOperations.FixedTimeEquals(plaintext,
decrypted))
                    throw new Exception("AES decrypt mismatch");
            }

            process.Refresh();
            var cpuAfter = process.TotalProcessorTime;

            var totalMs = encSw.Elapsed.TotalMilliseconds +
decSw.Elapsed.TotalMilliseconds;
            var cpuMs = (cpuAfter - cpuBefore).TotalMilliseconds;
            var cpuPercent = totalMs <= 0 ? 0 : Math.Min(100.0, (cpuMs /
totalMs) * 100.0);

            var ramDeltaMb = (process.WorkingSet64 - memBefore) / (1024.0 *
1024.0);
            if (ramDeltaMb < 0)
            {
                ramDeltaMb = process.WorkingSet64 / (1024.0 * 1024.0);
            }

            return new BenchmarkResult
            {
                AvgEncryptMs = encSw.Elapsed.TotalMilliseconds / iterations,
                AvgDecryptMs = decSw.Elapsed.TotalMilliseconds / iterations,
                CpuPercent = cpuPercent,
                RamMb = ramDeltaMb
            };
        }

        private static BenchmarkResult BenchmarkEcc(byte[] plaintext, int
iterations)
        {
            var process = Process.GetCurrentProcess();
            var cpuBefore = process.TotalProcessorTime;
            var memBefore = process.WorkingSet64;

            var encSw = new Stopwatch();
            var decSw = new Stopwatch();

            for (var i = 0; i < iterations; i++)
            {
                encSw.Start();
                var cipher = EncryptWithEcc(plaintext);
                encSw.Stop();

                decSw.Start();
                var decrypted = DecryptWithEcc(cipher);
                decSw.Stop();

                if (!CryptographicOperations.FixedTimeEquals(plaintext,
decrypted))
                {
                    throw new Exception("ECC decrypt mismatch");
                }
            }
```

```
            process.Refresh();
            var cpuAfter = process.TotalProcessorTime;

            var totalMs = encSw.Elapsed.TotalMilliseconds +
decSw.Elapsed.TotalMilliseconds;
            var cpuMs = (cpuAfter - cpuBefore).TotalMilliseconds;
            var cpuPercent = totalMs <= 0 ? 0 : Math.Min(100.0, (cpuMs /
totalMs) * 100.0);

            var ramDeltaMb = (process.WorkingSet64 - memBefore) / (1024.0 *
1024.0);
            if (ramDeltaMb < 0)
            {
                ramDeltaMb = process.WorkingSet64 / (1024.0 * 1024.0);
            }

            return new BenchmarkResult
            {
                AvgEncryptMs = encSw.Elapsed.TotalMilliseconds / iterations,
                AvgDecryptMs = decSw.Elapsed.TotalMilliseconds / iterations,
                CpuPercent = cpuPercent,
                RamMb = ramDeltaMb
            };
        }

        private static BenchmarkResult BenchmarkHybridAesEcc(byte[] plaintext,
int iterations)
        {
            var sessionKey = DeriveSessionKeyWithEcc();

            var process = Process.GetCurrentProcess();
            var cpuBefore = process.TotalProcessorTime;
            var memBefore = process.WorkingSet64;

            var encSw = new Stopwatch();
            var decSw = new Stopwatch();

            for (int i = 0; i < iterations; i++)
            {
                encSw.Start();
                var (cipher, iv) = EncryptAesCbc(plaintext, sessionKey);
                encSw.Stop();

                decSw.Start();
                var decrypted = DecryptAesCbc(cipher, sessionKey, iv);
                decSw.Stop();

                if (!CryptographicOperations.FixedTimeEquals(plaintext,
decrypted))
                {
                    throw new Exception("AES+ECC decrypt mismatch");
                }
            }

            process.Refresh();
            var cpuAfter = process.TotalProcessorTime;

            var totalMs = encSw.Elapsed.TotalMilliseconds +
decSw.Elapsed.TotalMilliseconds;
            var cpuMs = (cpuAfter - cpuBefore).TotalMilliseconds;
            var cpuPercent = totalMs <= 0 ? 0 : Math.Min(100.0, (cpuMs /
totalMs) * 100.0);

            var ramDeltaMb = (process.WorkingSet64 - memBefore) / (1024.0 *
```

```
1024.0);
            if (ramDeltaMb < 0)
            {
                ramDeltaMb = process.WorkingSet64 / (1024.0 * 1024.0);
            }

            return new BenchmarkResult
            {
                AvgEncryptMs = encSw.Elapsed.TotalMilliseconds / iterations,
                AvgDecryptMs = decSw.Elapsed.TotalMilliseconds / iterations,
                CpuPercent = cpuPercent,
                RamMb = ramDeltaMb
            };
        }

        private class BenchmarkResult
        {
            public double AvgEncryptMs { get; set; }
            public double AvgDecryptMs { get; set; }
            public double CpuPercent { get; set; }
            public double RamMb { get; set; }
        }

        private class EccCipher
        {
            public byte[] Ciphertext { get; set; } = [];
            public byte[] Iv { get; set; } = [];
            public byte[] EphemeralPublicKey { get; set; } = [];
        }

        private static byte[] GenerateRandomBytes(int size)
        {
            var data = new byte[size];
            RandomNumberGenerator.Fill(data);
            return data;
        }

        private static (byte[] Ciphertext, byte[] Iv) EncryptAesCbc(byte[]
plaintext, byte[] key)
        {
            using var aes = Aes.Create();
            aes.KeySize = 256;
            aes.Key = key;
            aes.Mode = CipherMode.CBC;
            aes.Padding = PaddingMode.PKCS7;
            aes.GenerateIV();

            using var encryptor = aes.CreateEncryptor();
            var cipher = encryptor.TransformFinalBlock(plaintext, 0,
plaintext.Length);
            return (cipher, aes.IV);
        }

        private static byte[] DecryptAesCbc(byte[] ciphertext, byte[] key,
byte[] iv)
        {
            using var aes = Aes.Create();
            aes.KeySize = 256;
            aes.Key = key;
            aes.IV = iv;
            aes.Mode = CipherMode.CBC;
            aes.Padding = PaddingMode.PKCS7;

            using var decryptor = aes.CreateDecryptor();
```

```
            return decryptor.TransformFinalBlock(ciphertext, 0,
ciphertext.Length);
        }

        private static EccCipher EncryptWithEcc(byte[] plaintext)
        {
            using var eph =
ECDiffieHellman.Create(ECCurve.NamedCurves.nistP256);

            using var serverPub = ECDiffieHellman.Create();
            serverPub.ImportSubjectPublicKeyInfo(ServerPublicKeyInfo, out _);

            var sharedSecret = eph.DeriveKeyMaterial(serverPub.PublicKey);
            var aesKey = SHA256.HashData(sharedSecret);

            var enc = EncryptAesCbc(plaintext, aesKey);

            var ephPublicInfo = eph.ExportSubjectPublicKeyInfo();

            return new EccCipher
            {
                Ciphertext = enc.Ciphertext,
                Iv = enc.Iv,
                EphemeralPublicKey = ephPublicInfo
            };
        }

        private static byte[] DecryptWithEcc(EccCipher cipher)
        {
            using var ephPub = ECDiffieHellman.Create();
            ephPub.ImportSubjectPublicKeyInfo(cipher.EphemeralPublicKey, out _);

            var sharedSecret = ServerEcc.DeriveKeyMaterial(ephPub.PublicKey);
            var aesKey = SHA256.HashData(sharedSecret);

            return DecryptAesCbc(cipher.Ciphertext, aesKey, cipher.Iv);
        }

        private static byte[] DeriveSessionKeyWithEcc()
        {
            using var client =
ECDiffieHellman.Create(ECCurve.NamedCurves.nistP256);
            using var server =
ECDiffieHellman.Create(ECCurve.NamedCurves.nistP256);

            var serverPubInfo = server.ExportSubjectPublicKeyInfo();

            using var serverPubForClient = ECDiffieHellman.Create();
            serverPubForClient.ImportSubjectPublicKeyInfo(serverPubInfo, out _);

            var secretClient =
client.DeriveKeyMaterial(serverPubForClient.PublicKey);

            return SHA256.HashData(secretClient);
        }
    }
}
```