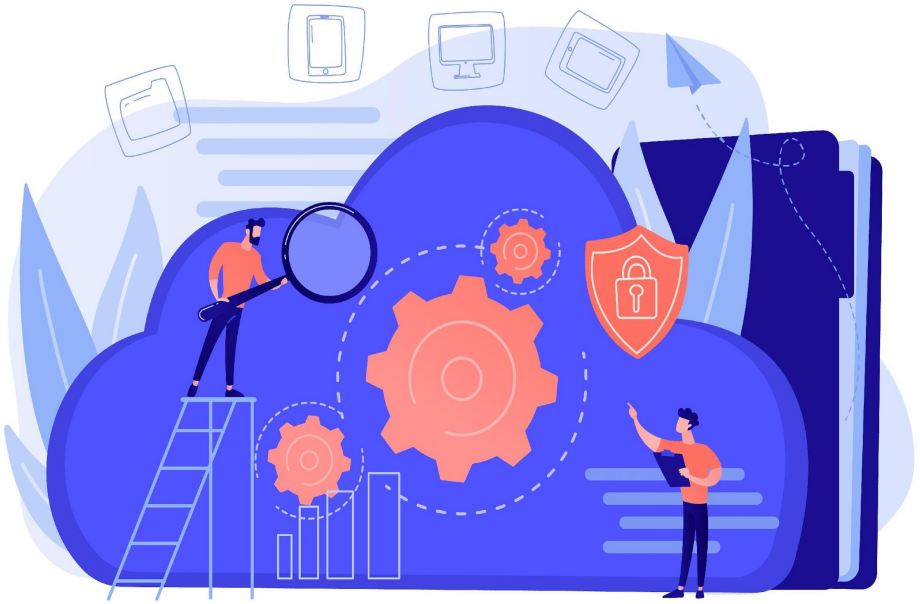
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Кафедра кібербезпеки та комп’ютерної інженерії

Система захищеного обміну повідомленнями в реальному часі (веб-додаток)



Здобувач: студент групи БІКСм-24 Чаюк С.О.
Керівник: Делембовський М.М.

Київ 2025 р.

Загальна характеристика роботи

Актуальність теми пов’язана з зростанням кіберзагроз та необхідності захисту даних, в умовах, коли браузери стають основним робочим середовищем.

Метою роботи є проектування архітектури та механізму шифрування в веб-середовищі для захищеного обміну даними, що базується на сучасних криптографічних стандартах.

Наукова новизна одержаних результатів полягає в адаптації та застосуванні комплексного поєднання криптографічних протоколів і стандартів для організації захищеного обміну даними в архітектурі веб-додатку.

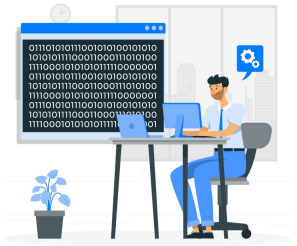
Об’єкт дослідження є процес захищеного обміну даними в реальному часі в клієнт-серверних веб-системах.

Предмет дослідження є крипто. методи, протоколи та архітектурні рішення для побудови веб-орієнтованих систем з наскрізним шифруванням.

Задачі, методи та структура дослідження

Задачі можна розділити на два основні блоки, а саме: аналіз та проектування.

Аналіз загроз безпеці обміну даних, протоколів E2EE та їхніх базових алгоритмів.
Обгрунтування вибору криптографічних примітивів для реалізації у браузері.



Проектування захищеної клієнт-серверної архітектури з мінімізацією довіри до сервера.
Розробка механізмів управління ключами та ядра шифрування.

Для вирішення поставлених задач у роботі використано комплекс методів: системний аналіз і синтез для огляду предметної області та порівняння архітектурних підходів; об’єктно-орієнтоване проектування для структурної декомпозиції системи та моделювання взаємодії ключових сутностей; прототипування для створення криптографічного ядра.

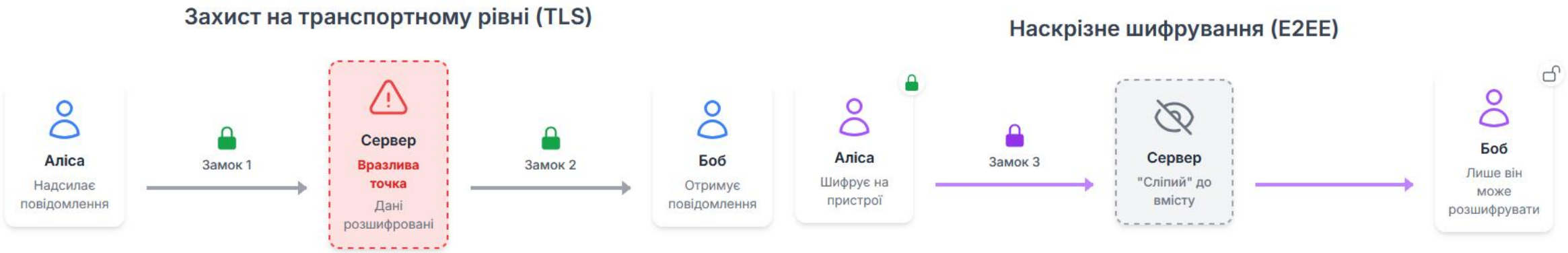
Робота складається зі вступу, 3 розділів, висновків, списку використаних джерел та додатків.

Аналіз загроз обміну даними

Будь-яка система, що працює з інформацією, традиційно оцінюється за трьома основними критеріями, відомими як “тріада CIA”.

Модель загроз визначає, від кого ми захищаємось. Для захищеного обміну даними ключовими є три типи супротивників:
- Пасивний супротивник
- Активний супротивник, Man-in-the-Middle
- Скомпрометований сервер

Для захисту від цих загроз використовуються дві принципово різні моделі: TLS проти E2EE.



Перший життєздатний протокол E2EE

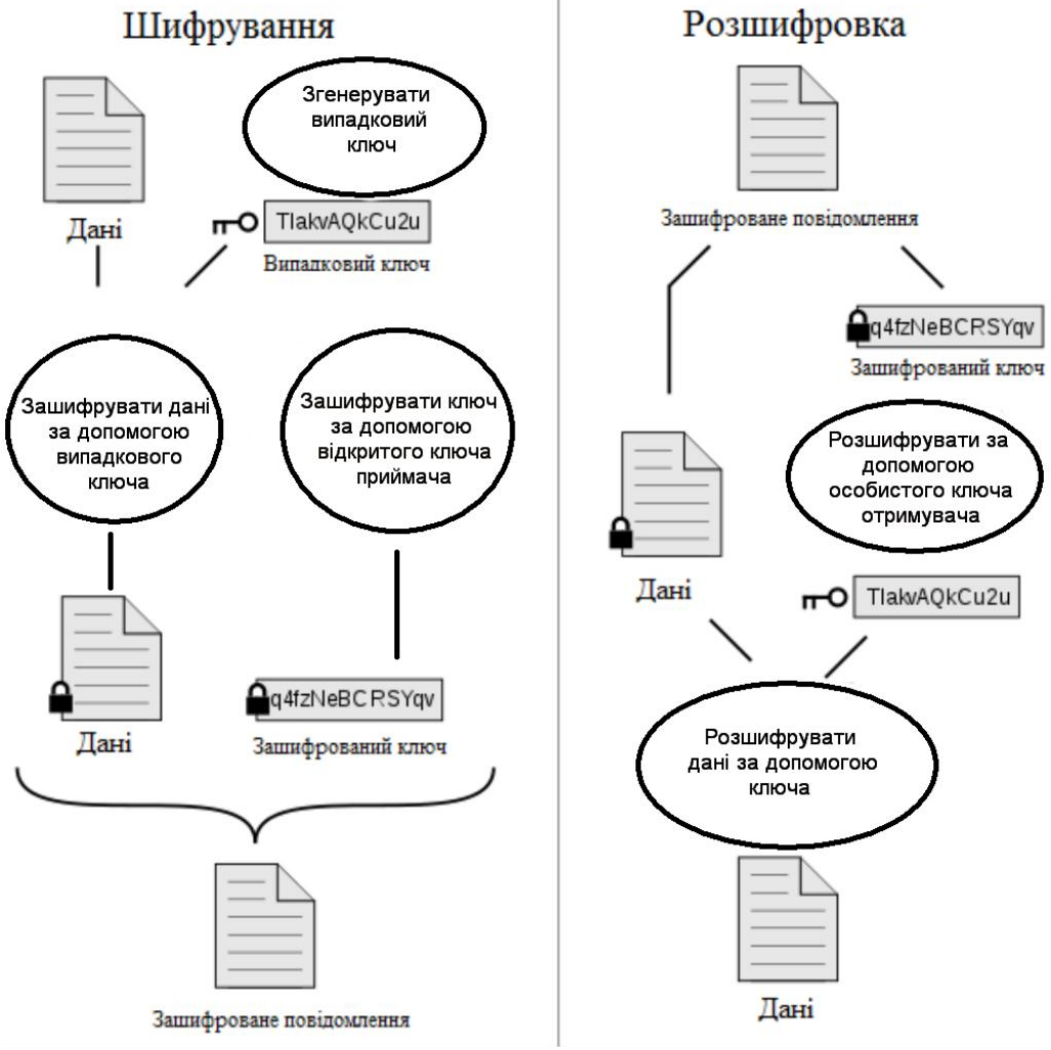
Першою широко розповсюдженою та криптографічно надійною системою E2EE був протокол Pretty Good Privacy.

Розроблений у 1990-х роках для захисту електронної пошти, PGP є класичним прикладом асинхронної криптографічної системи.

Головною вразливістю PGP є те, що довгостроковий приватний ключ є master key.

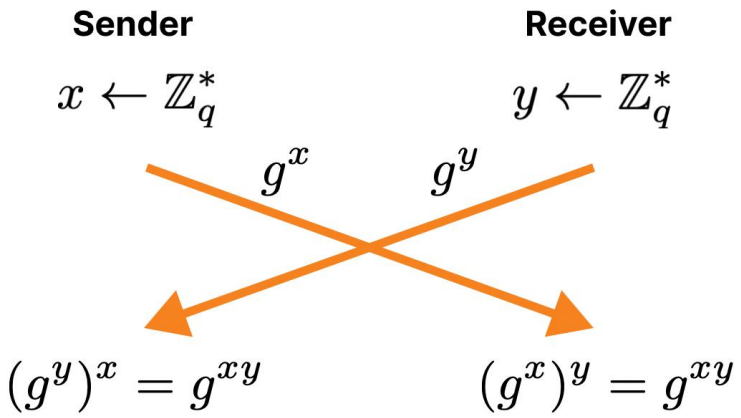
PGP вимагає від користувача ручного керування ключами. Аналогія — це як звичайні металеві ключі.

Також, PGP історично покладається на RSA. Це надійно, але математично дорого.

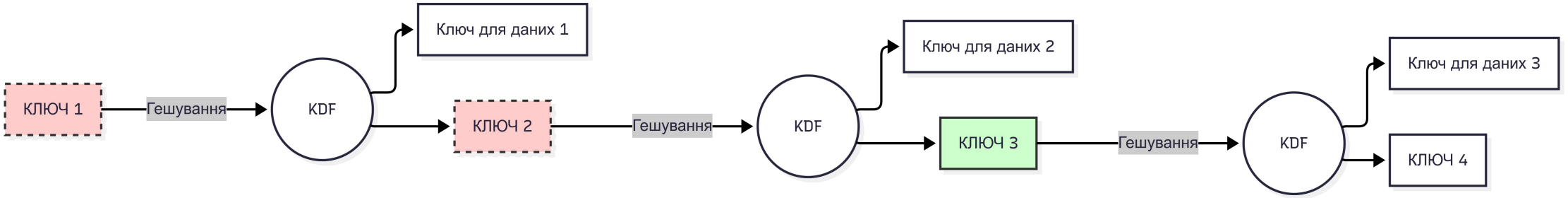


Протоколи OTR та SCIMP

У 2004 році було представлено протокол Off-the-Record, створений як цифровий аналог приватної розмови в зачиненій кімнаті. Його революційна особливість на момент виходу — впровадження Perfect Forward Secrecy. OTR не покладається на довгострокові ключі для шифрування. Натомість, на початку кожного сеансу він використовує протокол обміну ключами Діффі-Хеллмана для генерації нових, ефемерних сеансових ключів.



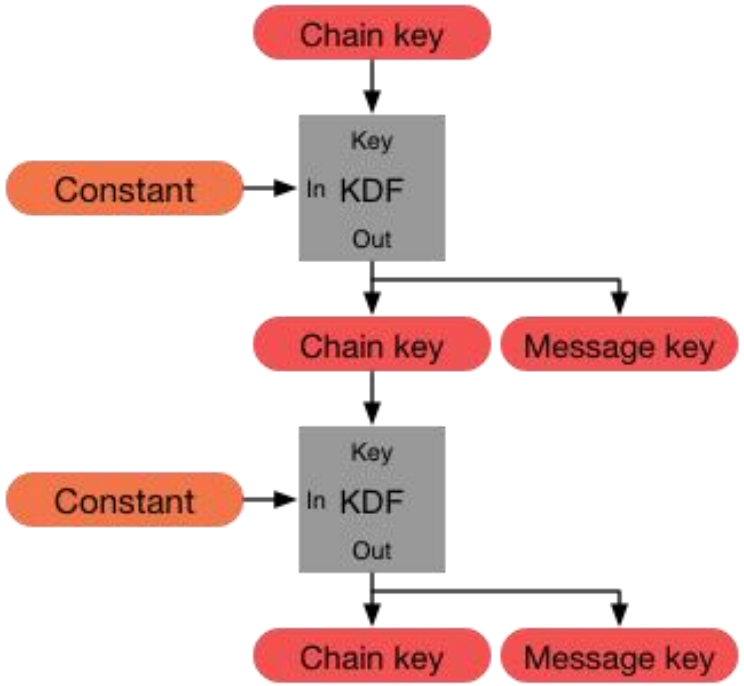
Розуміючи обмеженість OTR, розробники почали шукати шляхи для адаптації його властивостей до асинхронного середовища. Однією з таких спроб був Silent Circle Instant Messaging Protocol. Він використовує прогресивне шифрування (Ratcheting). Ключі оновлюються не просто на початку розмови, а з кожним новим повідомленням. Для генерації ключів застосовує KDF. Кожен наступний ключ математично вираховується з попереднього (через гешування), але зворотний процес неможливий.



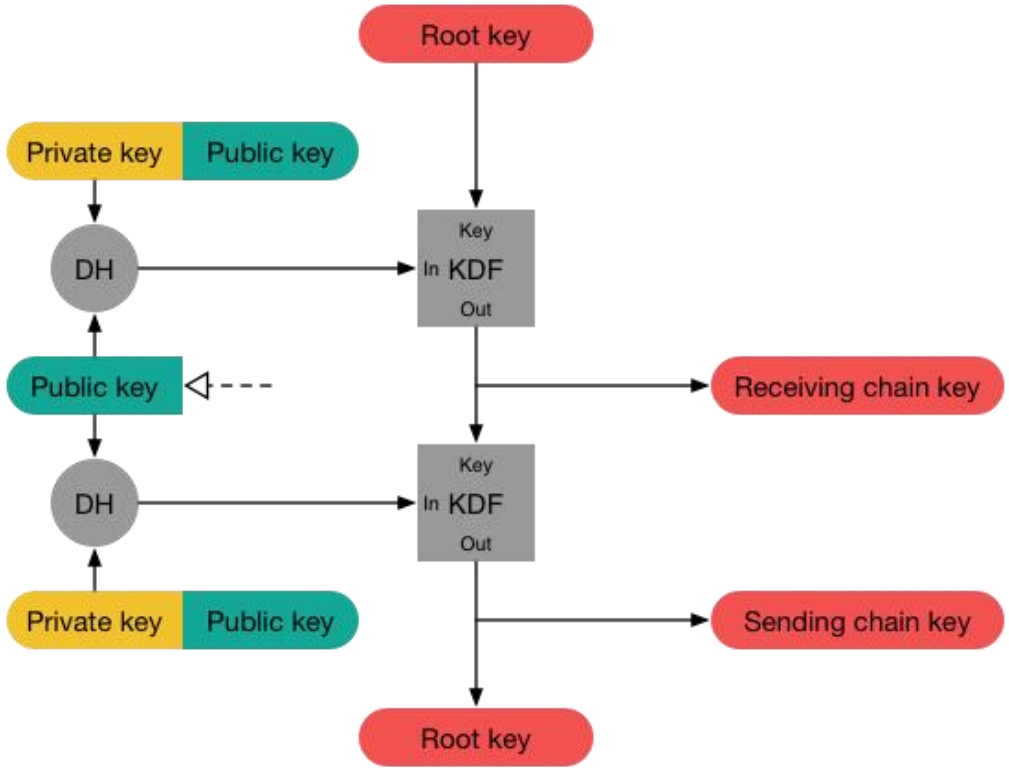
Signal Protocol

Сучасним “золотим стандартом” для комунікацій є протокол Signal, що поєднує механізм асинхронного встановлення сеансу X3DH та алгоритм подвійного храповика Double Ratchet. Архітектура використовує ефемерні тимчасові ключі для кожного сеансу та постійно оновлює їх за допомогою Double Ratchet. Ця комбінація забезпечує найвищі гарантії безпеки.

Symmetric-key ratchet



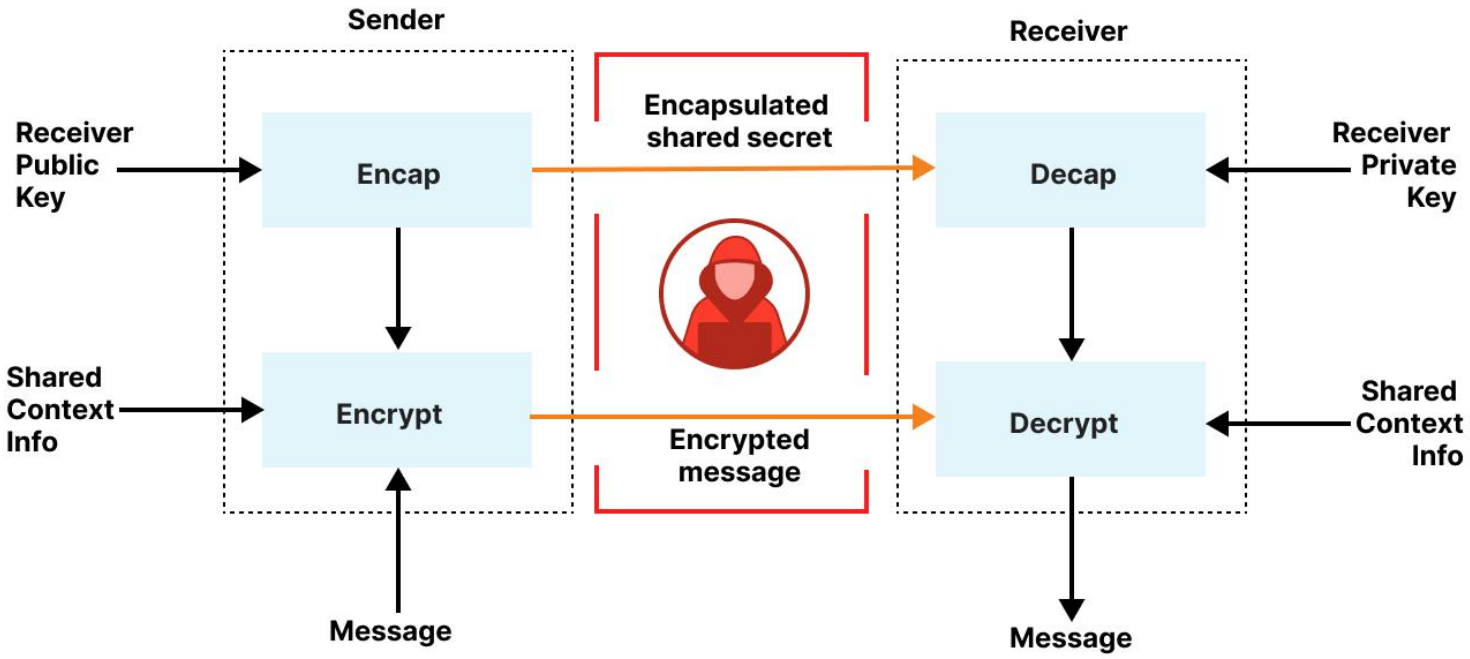
Diffie-Hellman ratchet



Стандартизація шифрування з відкритим ключем

Традиційне шифрування з відкритим ключем є повільним та має недоліки. Тому на практиці завжди використовується підхід, де генерується випадковий симетричний ключ, ним шифрується повідомлення, а потім цей симетричний ключ шифрується публічним ключем одержувача. Проблема полягала в тому, що існувало безліч способів це зробити, і багато з них були небезпечними. Hybrid Public Key Encryption, стандартизований як RFC 9180 в 2022 році, вирішує цю проблему, пропонуючи єдиний, безпечний та гнучкий стандарт для виконання цієї операції.

HPKE — це не протокол, а гнучкий конструктор, що стандартизує перевірений часом підхід, аналогічний PGP, але модернізований. Він чітко визначає комбінацію трьох примітивів: механізму інкапсуляції ключа KEM, функції виведення ключа KDF та схеми автентифікованого шифрування AEAD.



Генерація ключів та вибір еліптичної кривої

Основою системи захищеного обміну повідомленнями є криптографічна ідентичність користувача. У E2EE-системі ідентичність — це володіння парою асиметричних криптографічних ключів.

Процес генерації ключів відбувається виключно на стороні клієнта, в браузері, і базується на математичних властивостях еліптичних кривих над скінченними полями. Алгоритм генерації виглядає наступним чином:

- ✓ Генерація ентропії.
- ✓ Формування приватного ключа.
- ✓ Обчислення публічного ключа.

Операція скалярного множення на еліптичній кривій є “односторонньою функцією”: знаючи d і G, легко обчислити Q, але знаючи Q і G, обчислити d практично неможливо, адже проблема дискретного логарифмування.

Для реалізації механізму інкапсуляції ключів KEM еліптичну криву NIST P-256 (secp256r1). Цей вибір обгрунтований її широкою підтримкою у веб-браузерах через Web Cryptography API та відповідністю стандартам FIPS.

Отримані об’єкти CryptoKey зберігаються в оперативній пам’яті браузера.

Проблема зберігання приватного ключа

Після генерації ключів виникає критична архітектурна проблема: де і як безпечно зберігати приватний ключ identity key між сесіями? Це вимагає впровадження механізму експорту та відновлення ключа.

Використання мнемонічної фрази (Seed Phrase / BIP-39)

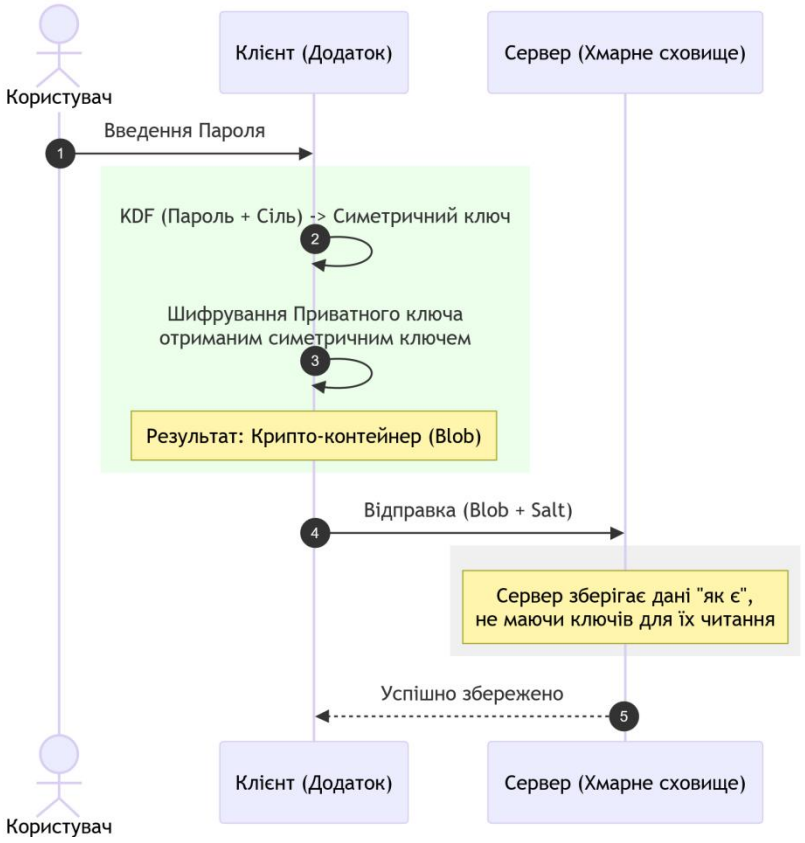
Метод, запозичений з криптовалютних гаманців, передбачає перетворення ентропії приватного ключа у читабельну послідовність з 12 або 24 слів згідно зі стандартом BIP-39. Система генерує слова, а користувач зобов’язаний фізично записати їх на папір або зберегти як Keystore File. Для входу на новому пристрої користувач вручну вводить ці слова, а система математично відновлює з них приватний ключ.

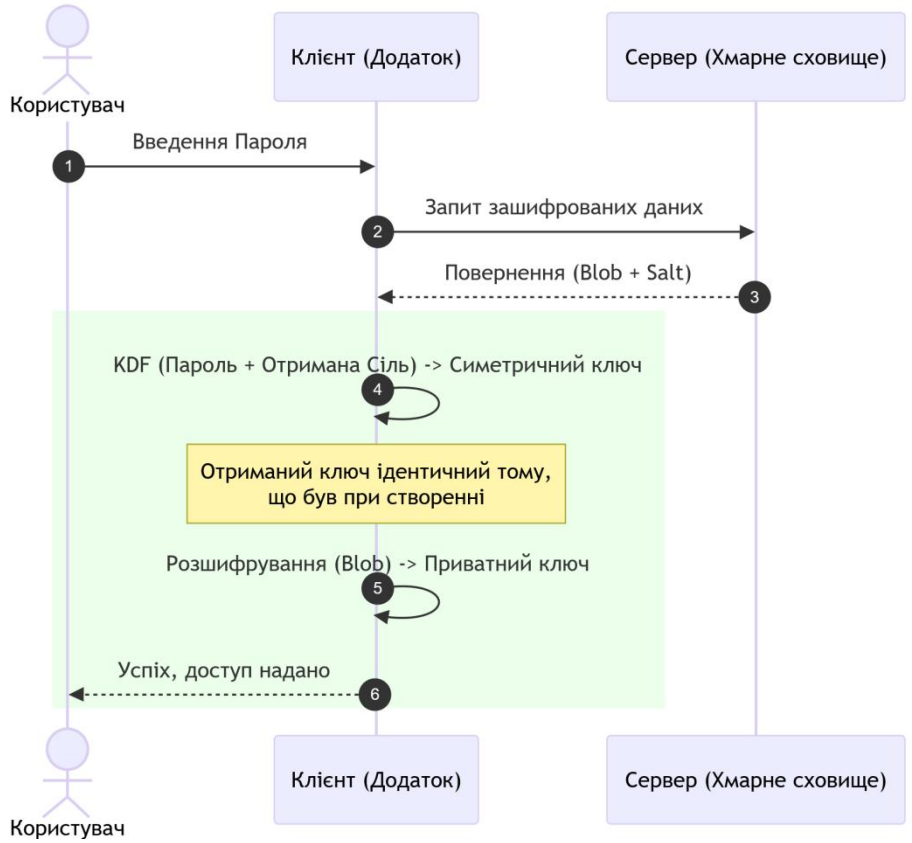
Прив’язка пристроїв через QR-код

У цьому підході приватний ключ ніколи не покидає пристрій, на якому був створений, у довгостроковій перспективі. Новий пристрій, наприклад десктоп, генерує власну ефемерну пару ключів і показує QR-код. Основний пристрій, де користувач вже автентифікований, сканує код і через захищений канал передає зашифровану копію свого ключа ідентичності.

Серверне зберігання зашифрованого приватного ключа

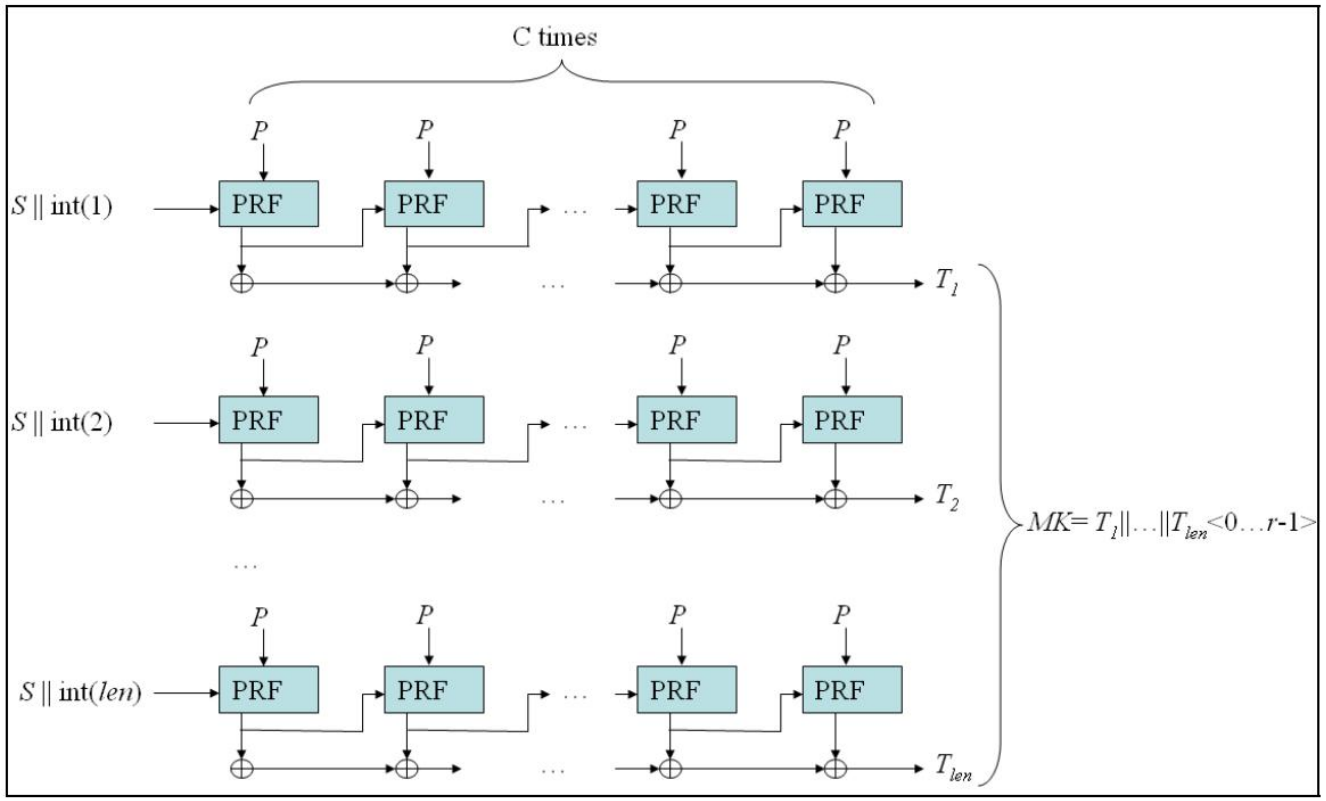
Підхід, який є компромісом між зручністю хмарної синхронізації та безпекою Zero-Knowledge. Він передбачає зберігання приватного ключа на сервері, але у вигляді криптографічного контейнера, зашифрованого ключем, похідним від пароля користувача. В KDF застосовується алгоритм деривації ключа PBKDF2. Шифрування контейнера виконується алгоритмом AES-GCM.





Password-Based Key Derivation Function 2

PBKDF2 — це стандарт NIST, обраний для перетворення пароля користувача на майстер-ключ. Його вибір замість простого гешу, як SHA-256, є критичним. PBKDF2 розроблений для протидії атакам перебору brute-force завдяки двом механізмам: сіль (salt) та ітерації, за якими функція виконує гешування



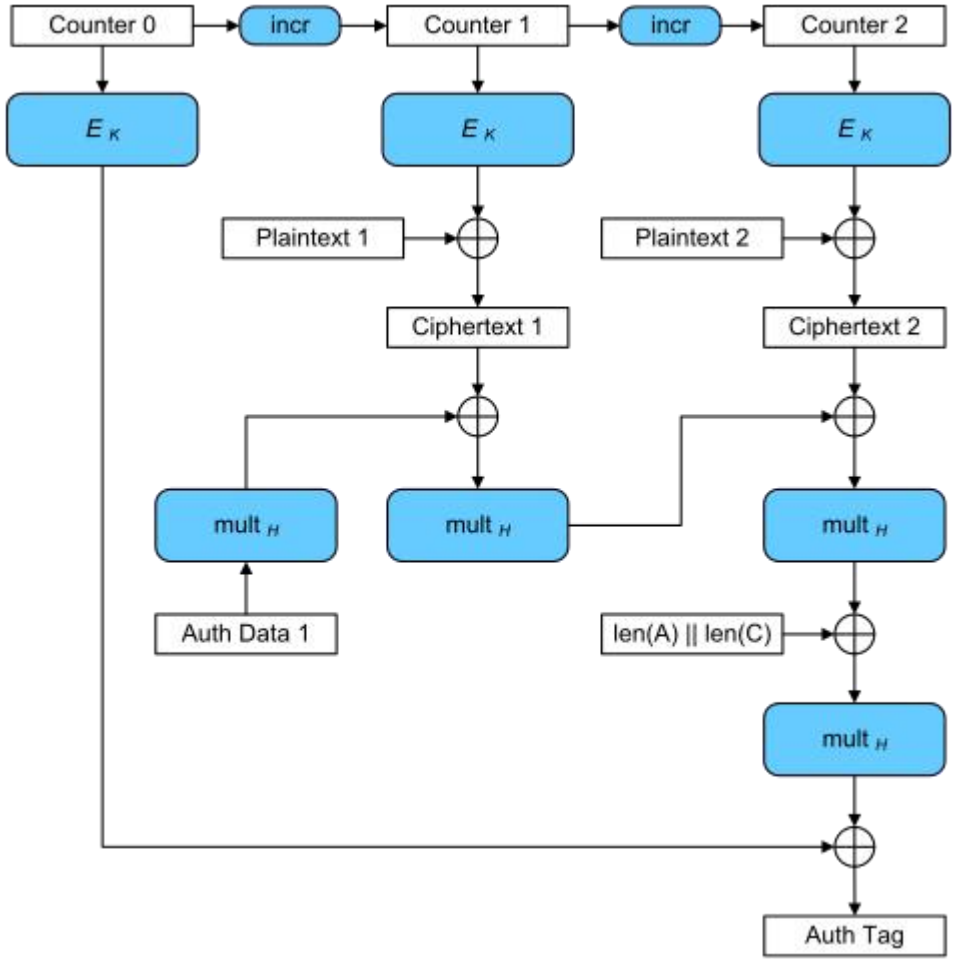
Симетричне шифрування з AES-GCM

AES-GCM (Galois/Counter Mode) обраний для шифрування приватного ключа користувача і можливо повідомлень за допомогою майстер-ключа, отриманого з PBKDF2. Його вибір також не є випадковим.

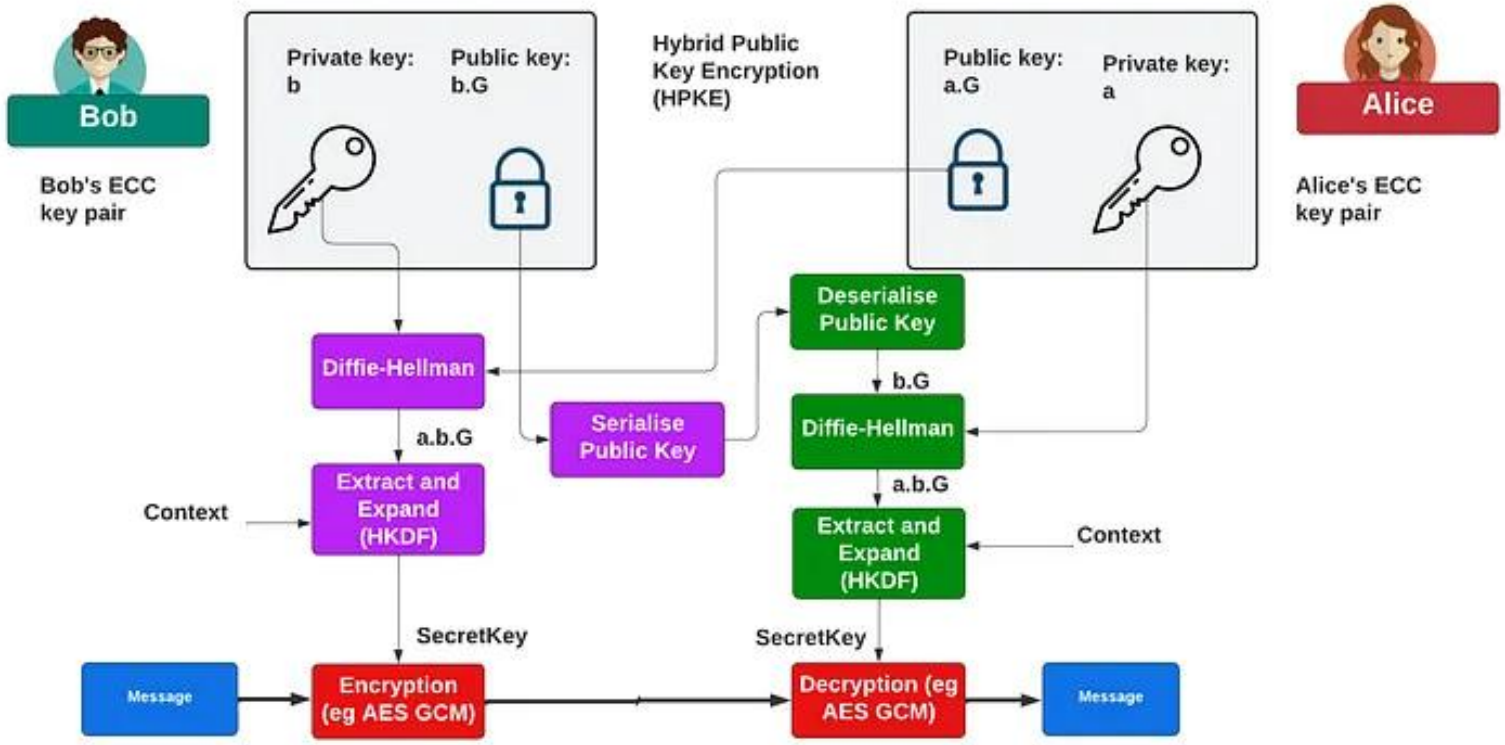
GCM поєднує AES з режимом лічильника для шифрування та режимом Галуа для автентифікації. Це забезпечує як конфіденційність, так і цілісність даних.

У AES/GCM режим лічильника CTR перетворює блоковий шифр у потоковий шифр шляхом шифрування послідовних значень лічильника. Цей метод дозволяє здійснювати паралельну обробку та ефективні операції шифрування/дешифрування.

Режим Galois забезпечує автентифікацію повідомлень шляхом генерації тегу, який перевіряє справжність даних. Це гарантує, що дані не були підроблені.



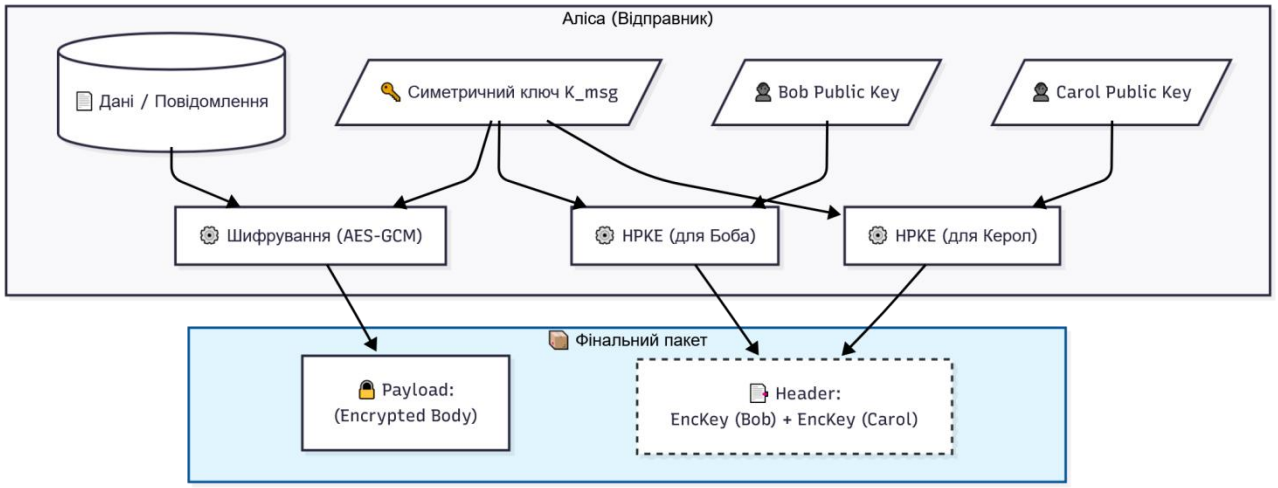
Базовий механізм шифрування через HPKE

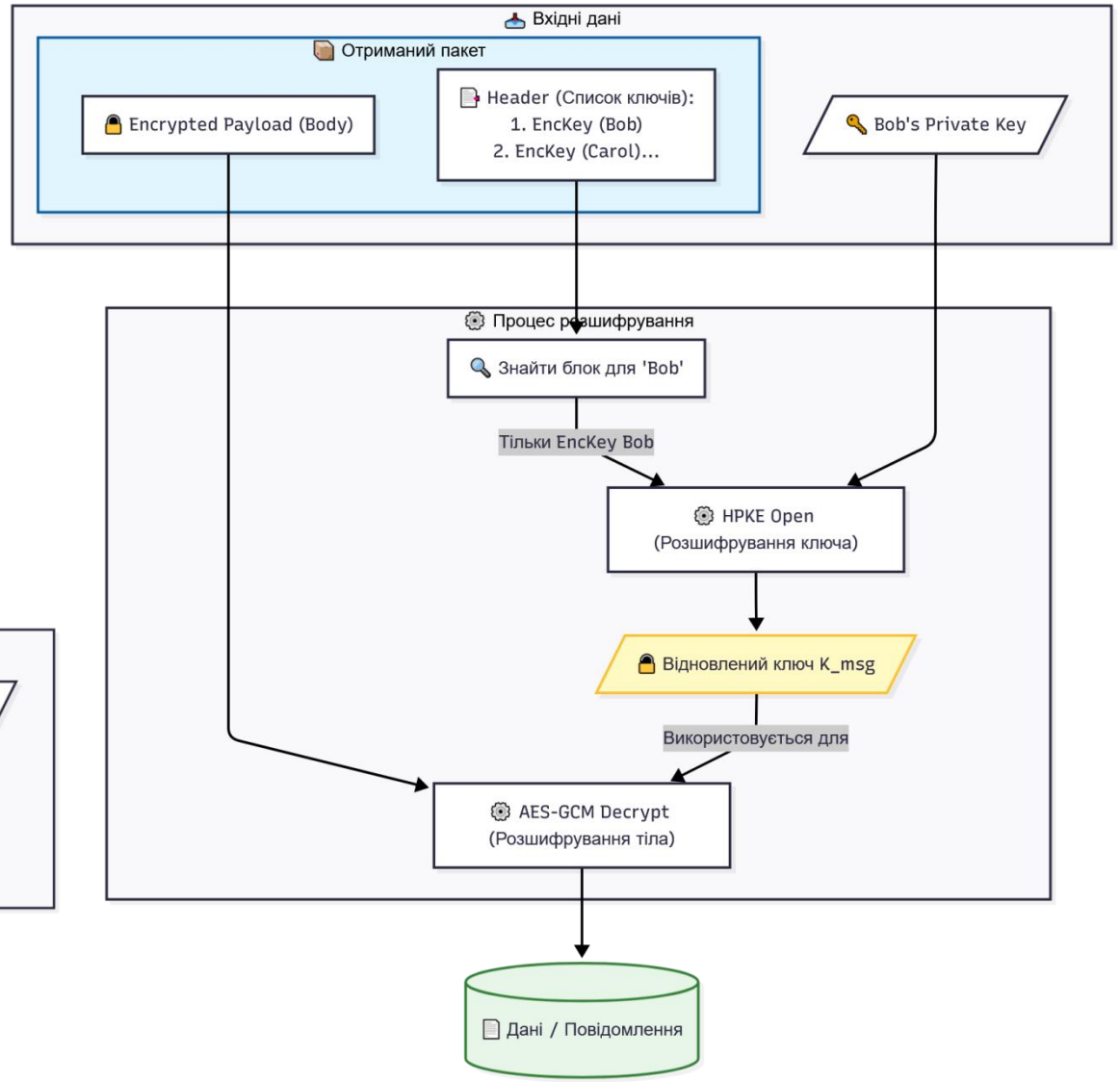


Цей підхід є криптографічно стійким, але він ефективний лише для двох учасників. Оскільки спільний секрет, а отже, і контекст, унікально прив’язаний до пари ключів учасників, користувач Б не може використати той самий шифротекст для третього учасника. Для кожного отримувача процедуру шифрування доведеться виконувати заново, що утворює дублювання даних.

Гібридне шифрування з AES-GCM і HPKE

Для вирішення проблеми масштабування було застосовано патерн Digital Envelope. Ідея полягає у розділенні шифрування даних і шифрування ключів. У цій схемі ми вводимо поняття симетричного сеансового ключа повідомлення. Це дозволяє серверу зберігати лише одну копію зашифрованих даних, незалежно від кількості отримувачів, що економить дисковий простір та трафік.



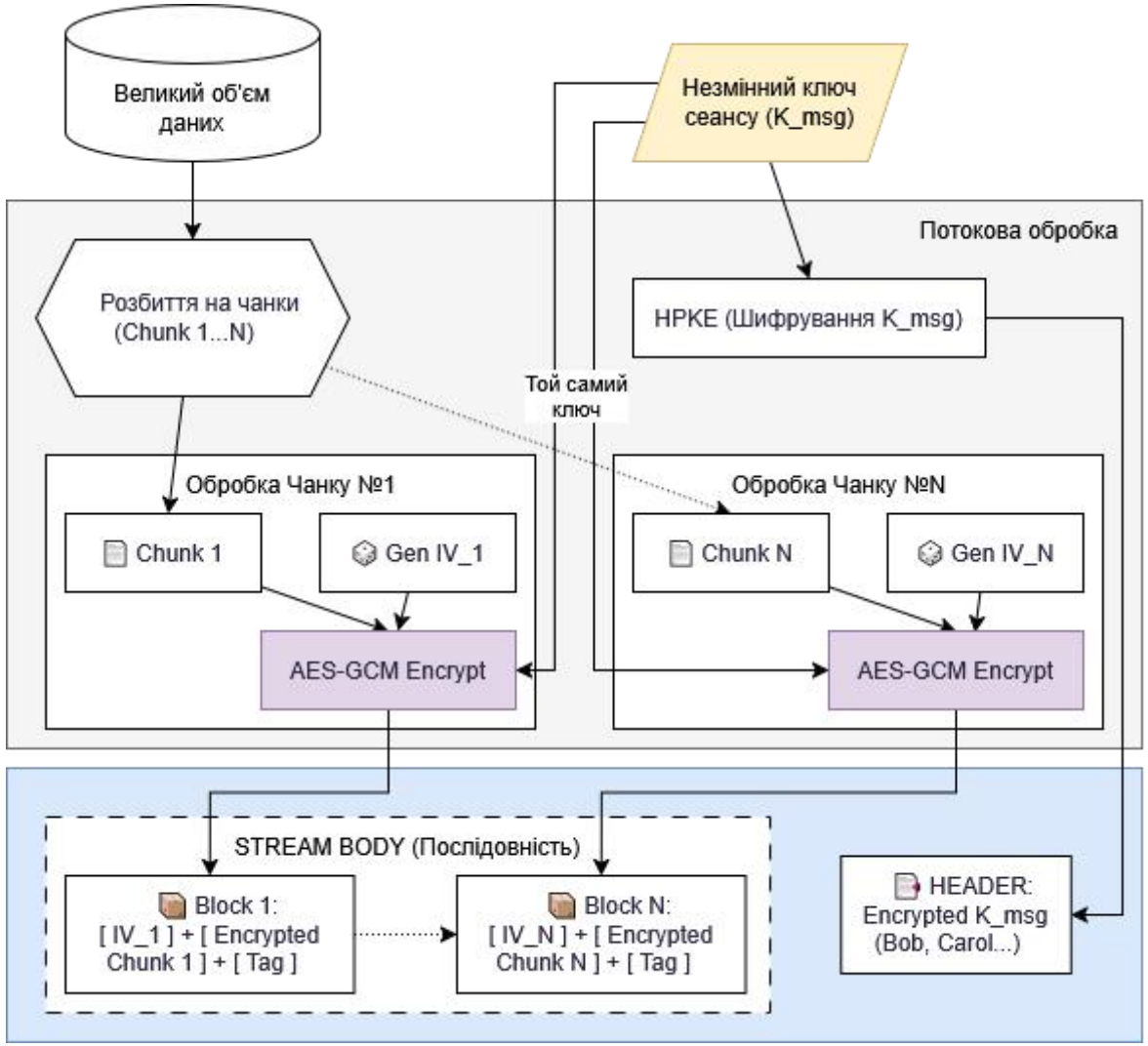


Оптимізація обробки великих масивів даних

Одним із критичних викликів при реалізації наскрізного шифрування у веб-середовищі є обмеження ресурсів клієнтського пристрою, зокрема оперативної пам’яті. Стандартна реалізація методів Web Cryptography API, працює за принципом “все або нічого”: для виконання операції весь масив даних має бути завантажений в пам’ять у вигляді об’єкта ArrayBuffer.

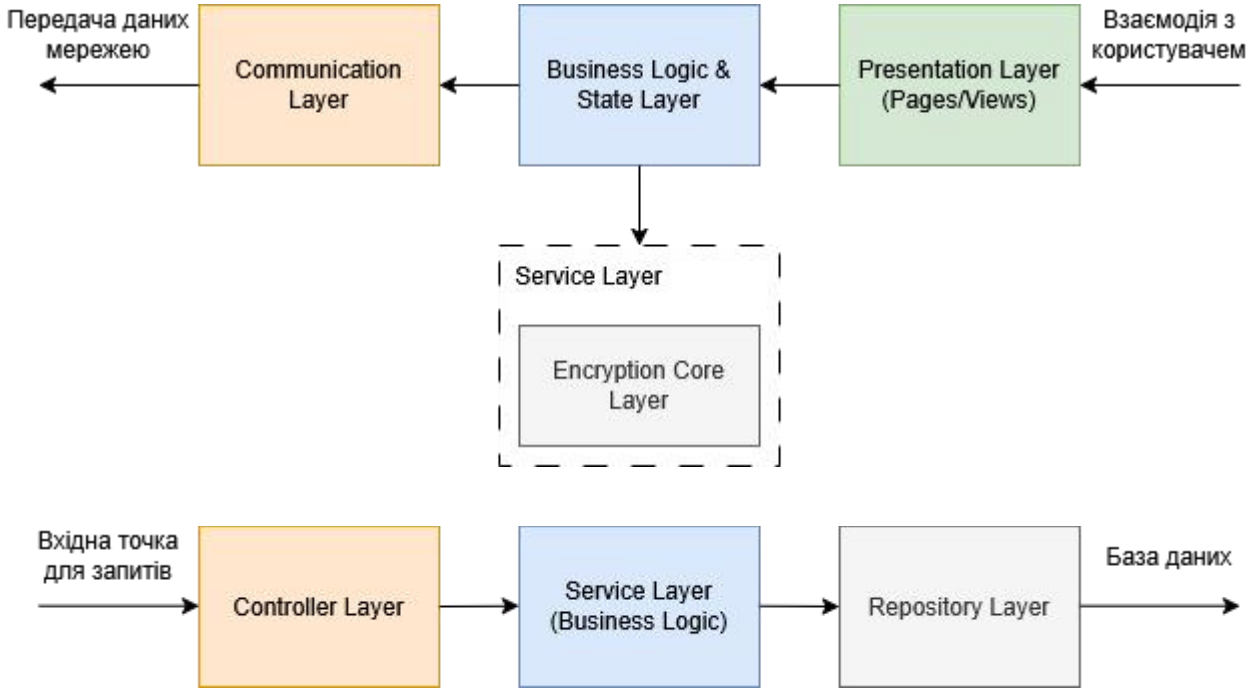
Для вирішення цих проблем було розроблено алгоритм потокового шифрування, який розширює спосіб використання алгоритму гібридного шифрування.

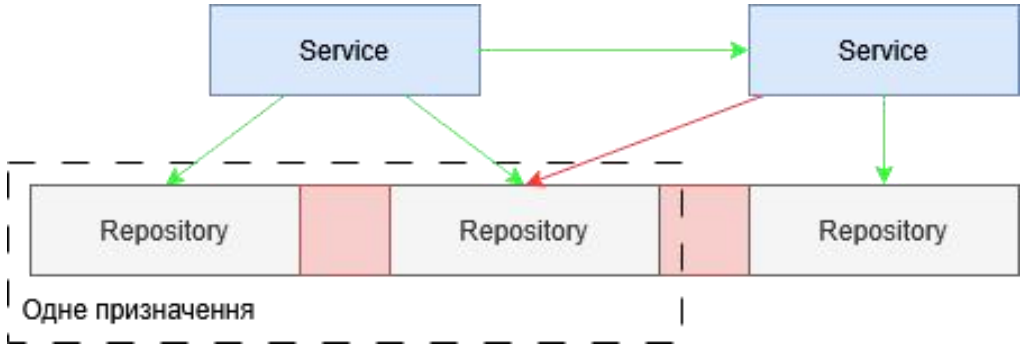
Такий підхід дозволяє обробляти масиви даних будь-якого розміру використовуючи вже розроблене гібридне шифрування і не змінюючи його архітектуру.

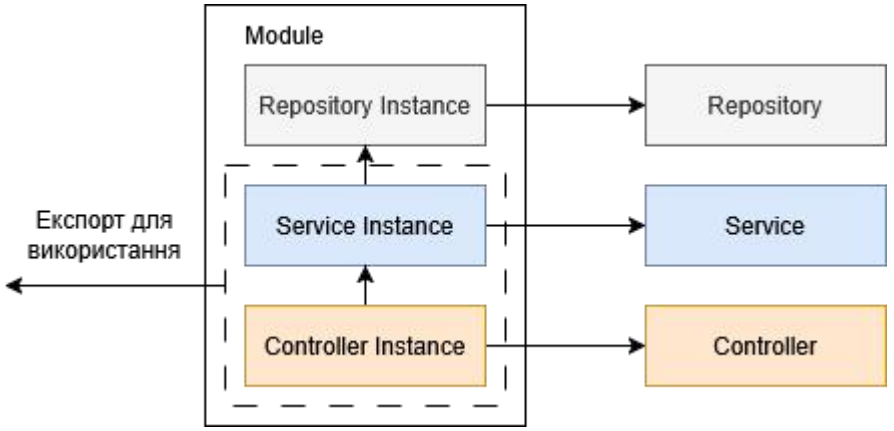


Загальна архітектура клієнт-серверної системи

Для впровадження механізмів шифрування і їх ефективного використання в веб-системах необхідно забезпечити надійність, масштабованість та легку підтримку. Тому, архітектура спроектована на основі принципів слабкої зв’язності компонентів, чіткого розмежування відповідальності та розшарування. Такий підхід дозволяє ізолювати логіку шифрування від інтерфейсу користувача та відокремити бізнес-правила від деталей реалізації.

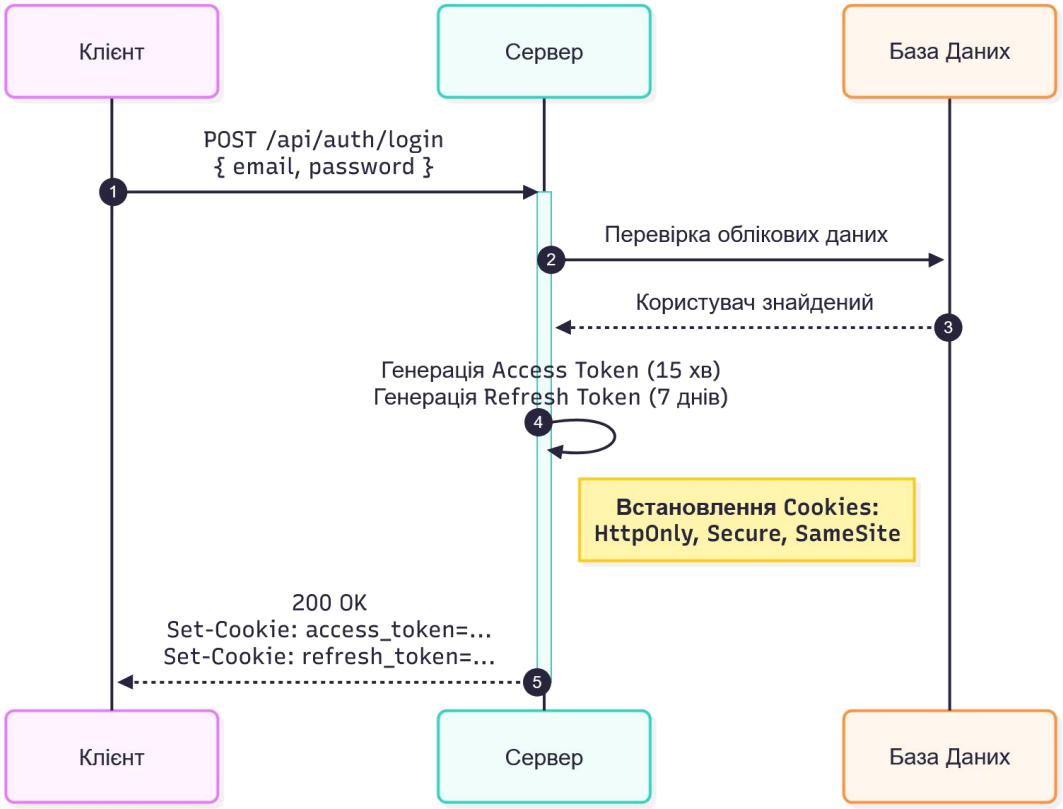


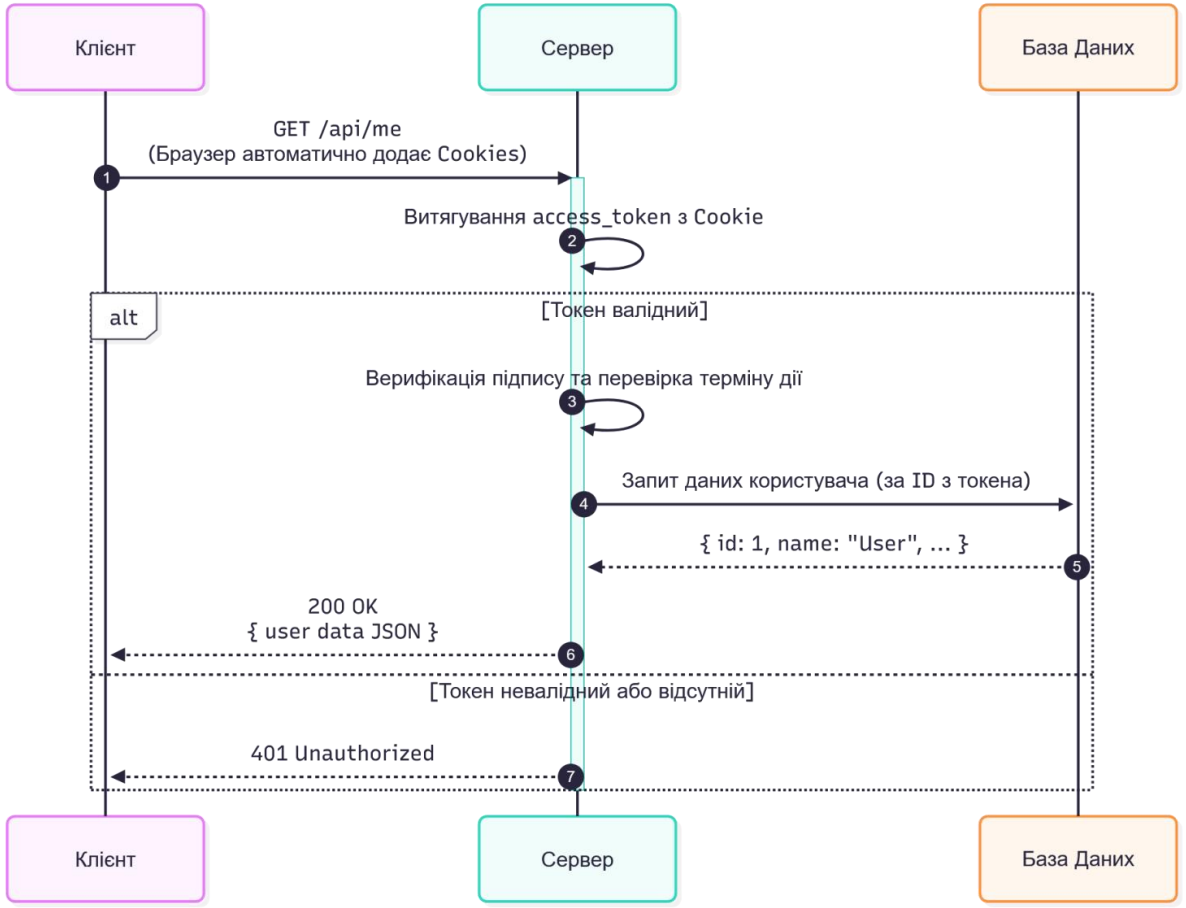




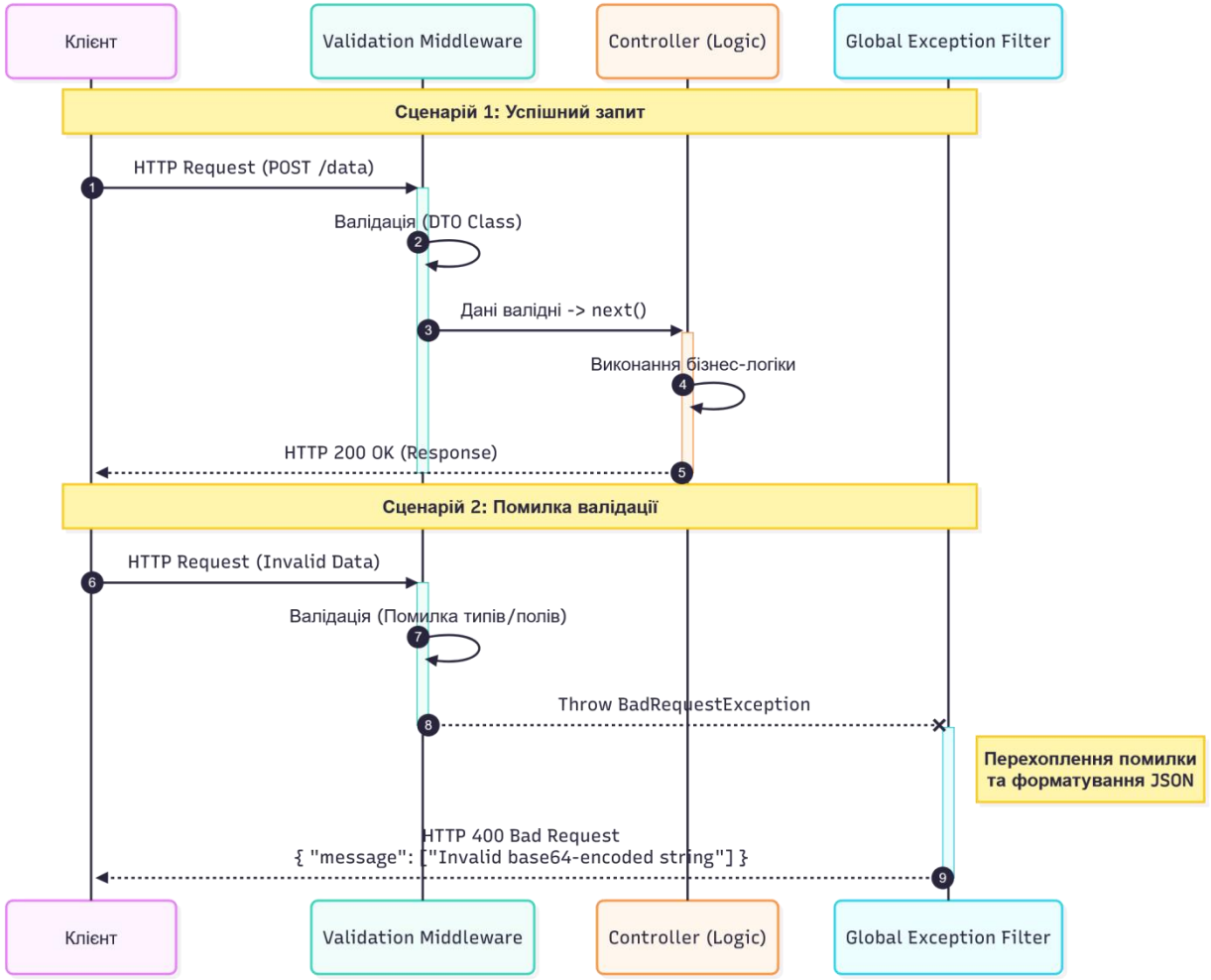
Механізми авторизації та безпеки сесій

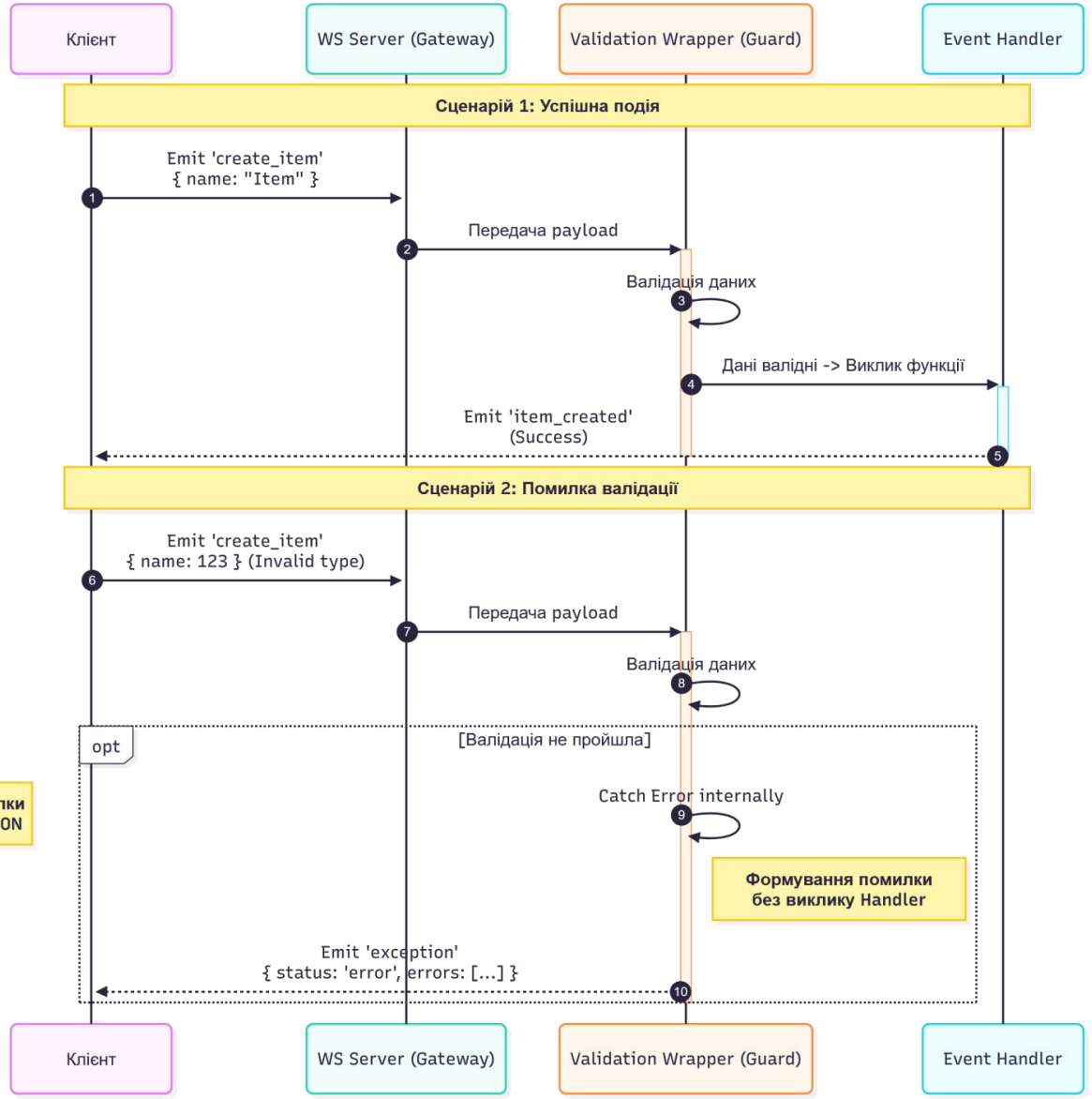
Ключовим елементом архітектури є система ідентифікації та управління сесіями користувачів. Оскільки додаток оперує конфіденційними даними, навіть у зашифрованому вигляді, до механізмів авторизації висуваються підвищені вимоги безпеки.





Обмін даним та стандартизація взаємодії





Зберігання криптографічних артефактів

Центральним елементом серверної інфраструктури є база даних, яка забезпечує персистентне зберігання стану системи. У контексті захищеної системи обміну повідомленнями до бази даних висуваються специфічні вимоги. Вона повинна виступати надійним сховищем не лише для реляційних зв’язків, але й для криптографічних артефактів: ключів, шифротекстів, векторів ініціалізації.

Ключовою є вимога до формату зберігання даних.

В якості системи управління базами даних обрано об’єктно-реляційну СУБД PostgreSQL, а для вирішення проблем ручного управління SQL-запитами та забезпечення type safety на рівні доступу до даних, Prisma ORM.

Для забезпечення функціонування системи захищеного обміну даними було розроблено реляційну схему даних, яка включає різні моделі моделі.

schema.prisma

```
model User {
  id
  username
  email
  hashedPassword
  avatarName

  // E2EE Data
  publicKey
  encryptedPrivateKey
  kdfSalt
  encryptionIV
}
```

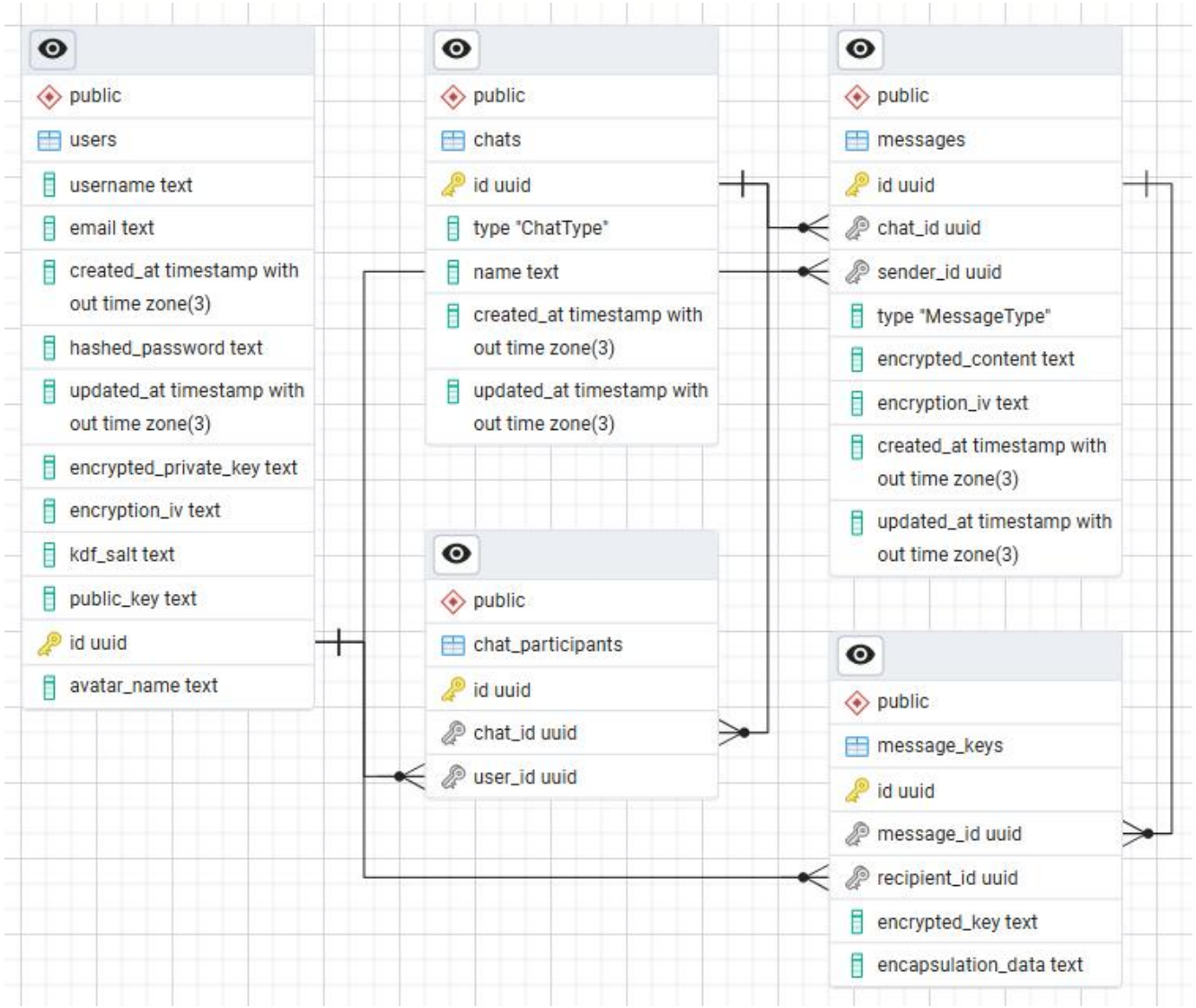
```
model Chat {
  id
  type
  name
  participants
}

model Message {
  id
  chatId
  senderId
  type

  // E2EE Data
  encryptedContent
  encryptionIV
}
```

```
model ChatParticipant {
  id
  chatId
  userId
}

model MessageKey {
  id
  messageId
  recipientId

  // E2EE Data
  encryptedKey
  encapsulationData
}
```

Міграція та візуалізація схеми

На основі описаної схеми в schema.prisma за допомогою команди prisma migrate генеруються SQL-міграції, які створюють відповідні таблиці, індекси та зовнішні ключі у PostgreSQL. Це гарантує повну відповідність структури бази даних коду додатку.

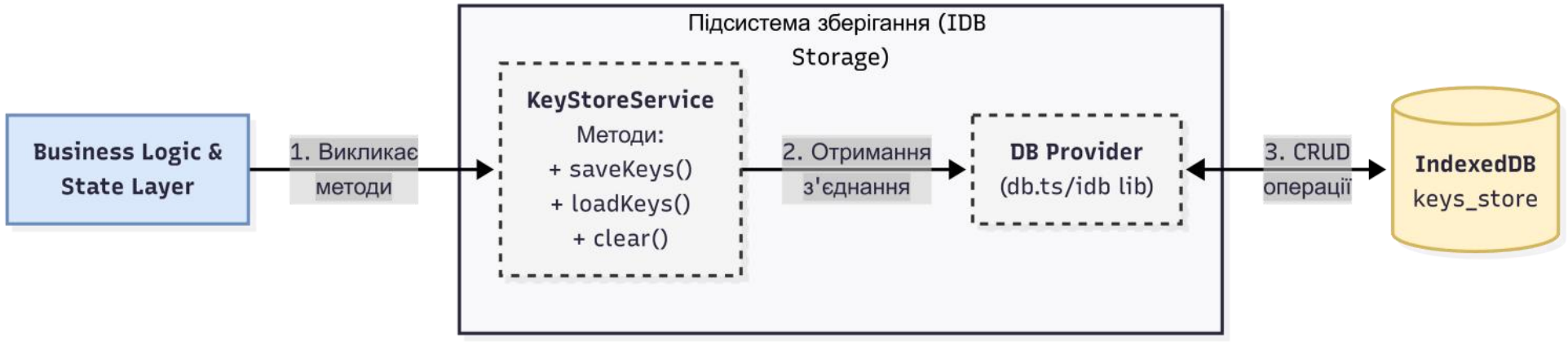
Запропонована архітектура є гнучкою та легко адаптується для підтримки обміну файлами.



Організація локального сховища ключів

Критичною умовою зручності використання E2EE-додатку є збереження криптографічної сесії між перезавантаженнями сторінки. Користувач не повинен проходити процедуру деривації ключів при кожному оновленні вкладки браузера. Для вирішення цієї задачі було спроектовано шар локальної персистентності.

Стандартний механізм localStorage виявився непридатним для зберігання криптографічних ключів, бо працює виключно з рядками. Натомість було обрано IndexedDB



Управління життєвим циклом криптографічної ідентичності

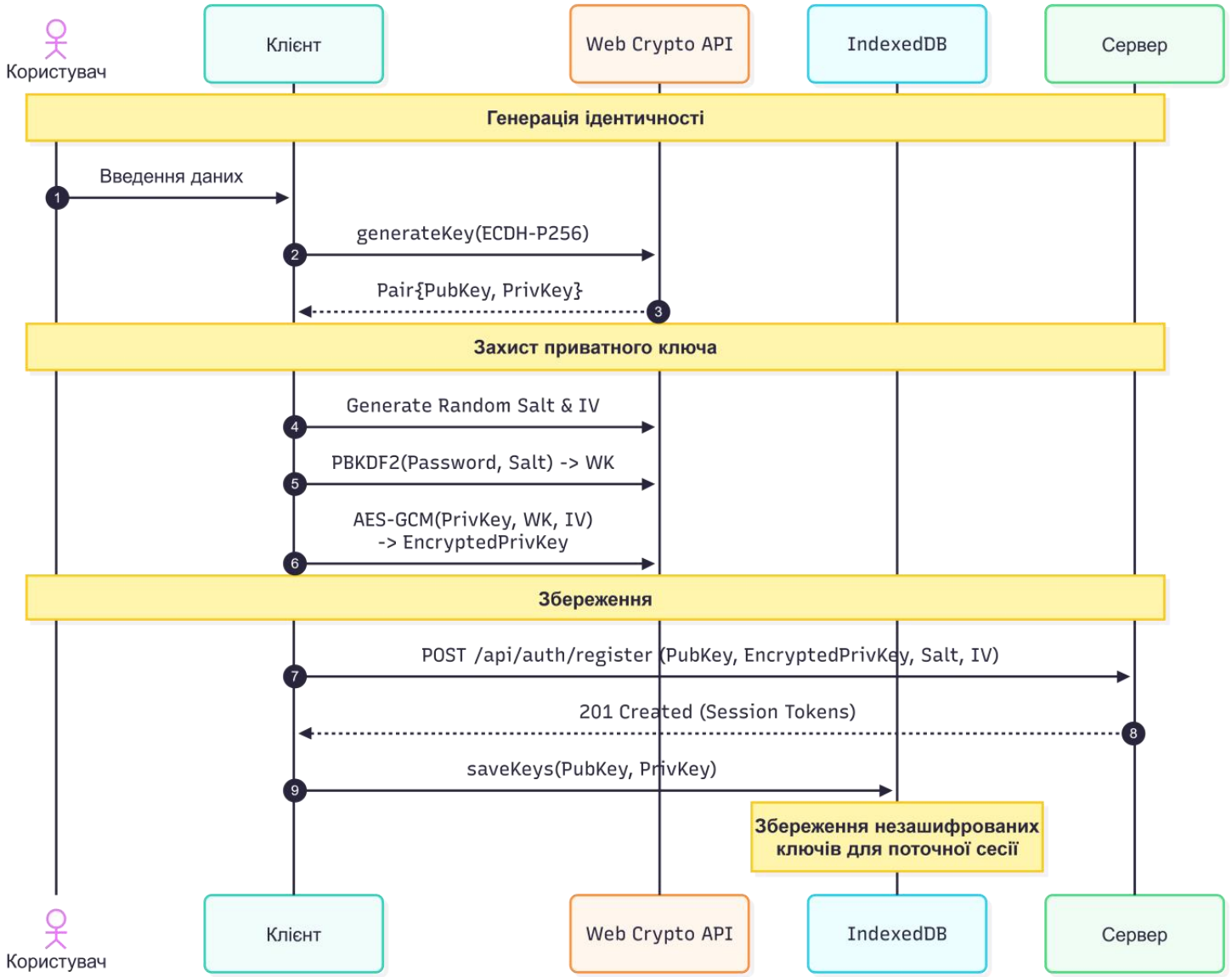
Ефективне управління криптографічними ключами є критично важливим аспектом системи наскрізного шифрування, оскільки втрата приватного ключа означає безповоротну втрату доступу до історії, а його компрометація — порушення конфіденційності. Життєвий цикл ідентичності в розробленій системі побудований на взаємодії трьох компонентів:

- оперативної пам’яті клієнта, де ключі використовуються,
- локального сховища IndexedDB, де ключі зберігаються між перезавантаженнями
- сервера, де зберігається зашифрована копія.

Розглядаємо детальний опис алгоритмів для трьох основних сценаріїв використання: реєстрації, входу в систему та відновлення сесії.

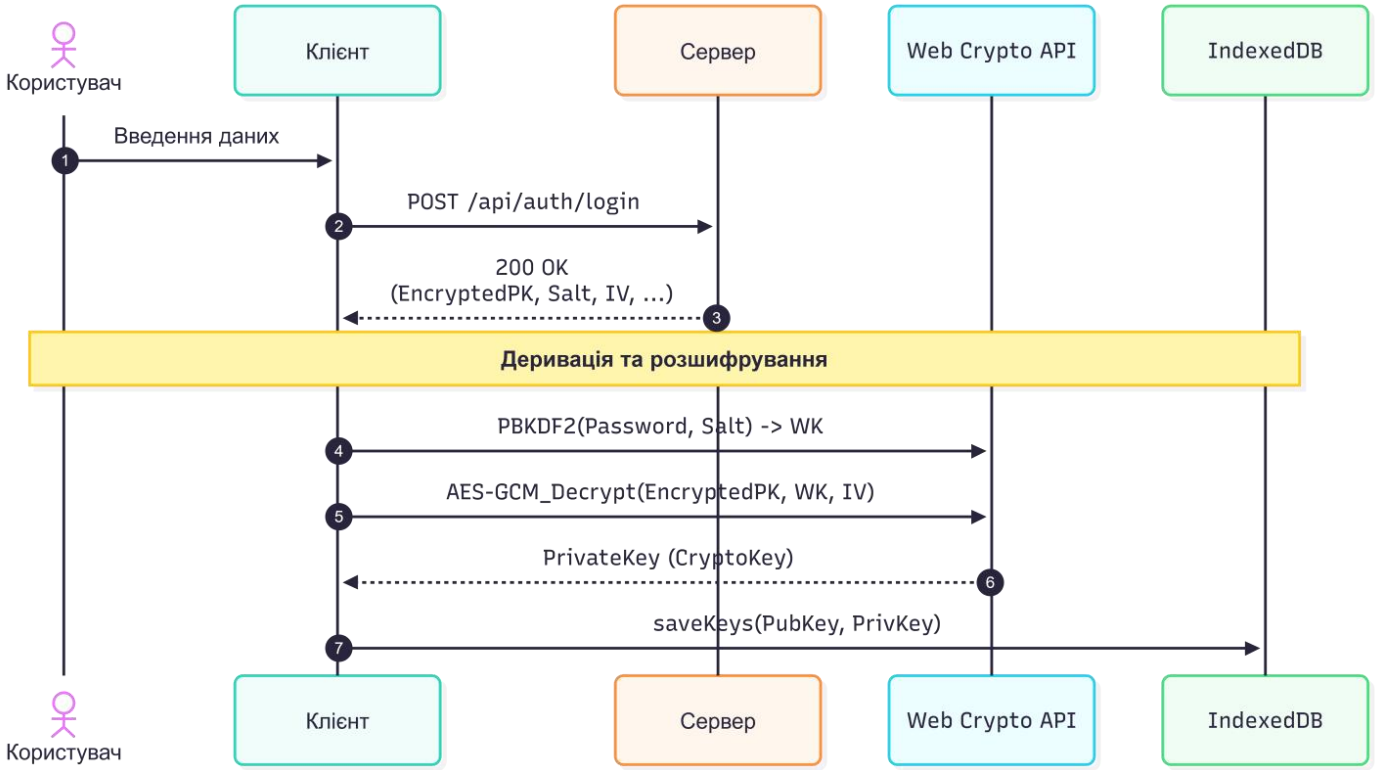
Послідовність процесу реєстрації та первинної генерації ключів

Процес реєстрації нового користувача виходить за межі простого створення запису в базі даних. Він включає генерацію криптографічного матеріалу та створення механізму його відновлення.



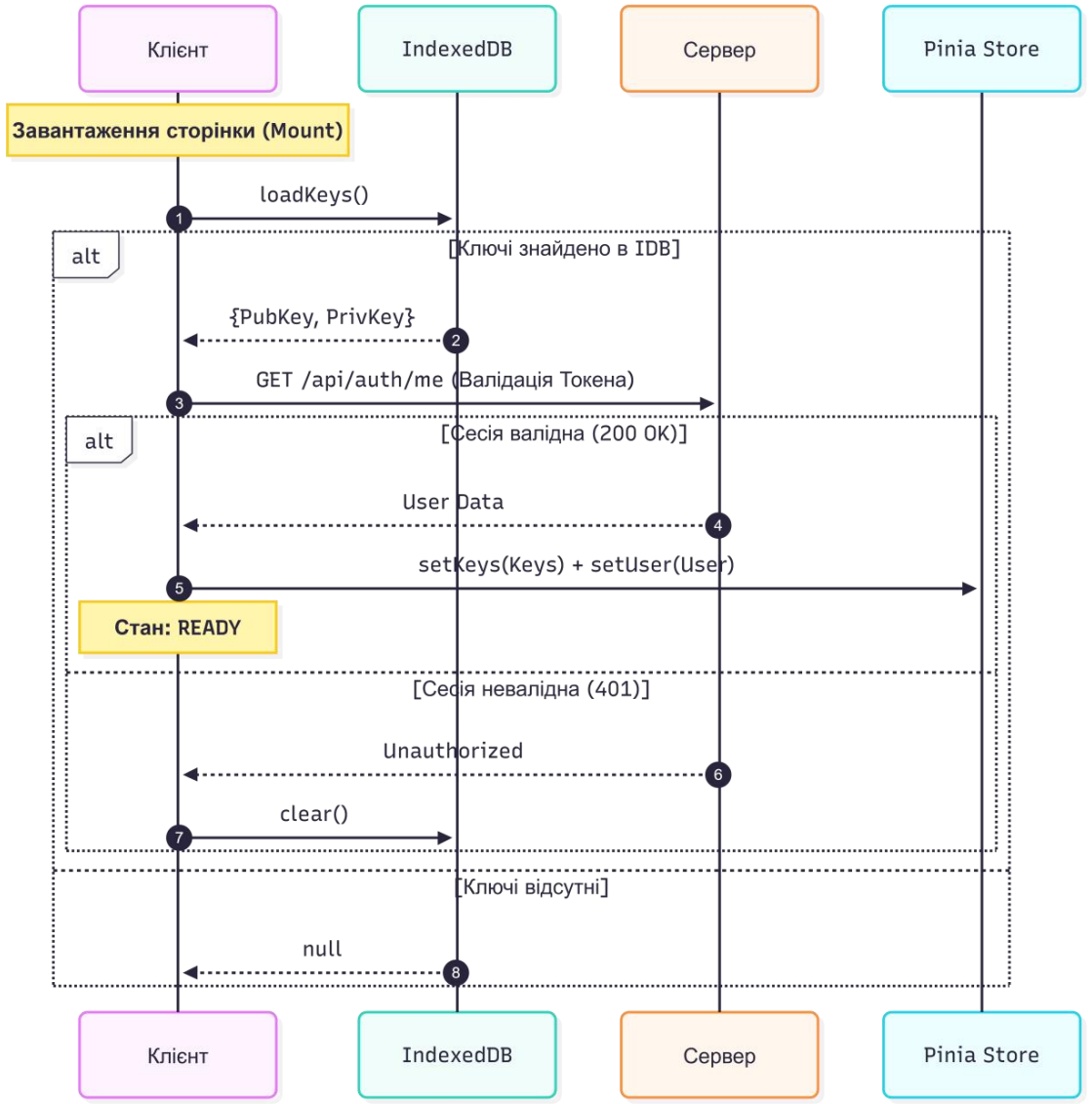
Послідовність процесу входу та відновлення ключів з сервера

Відновлення ключів з сервера активується, коли користувач входить у систему з нового пристрою або після очищення даних браузера. Головна задача — безпечно завантажити та розшифрувати приватний ключ.



Послідовність процесу відновлення сесії

Відновлення сесії відбувається при кожному оновленні сторінки або повторному відкритті вкладки. Система повинна відновити готовність до шифрування/розшифрування максимально швидко і непомітно для користувача, але важливо переконатися у валідності поточної сесії на сервері.

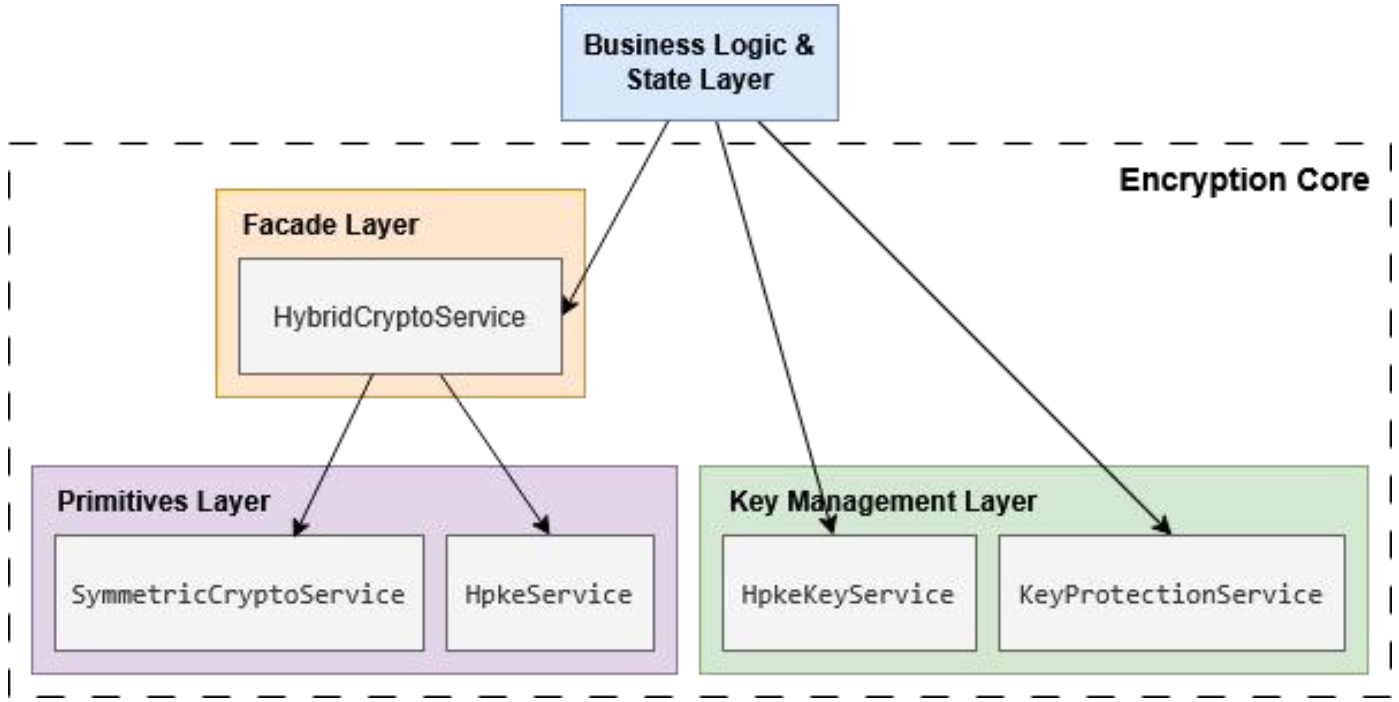


Реалізація ядра шифрування

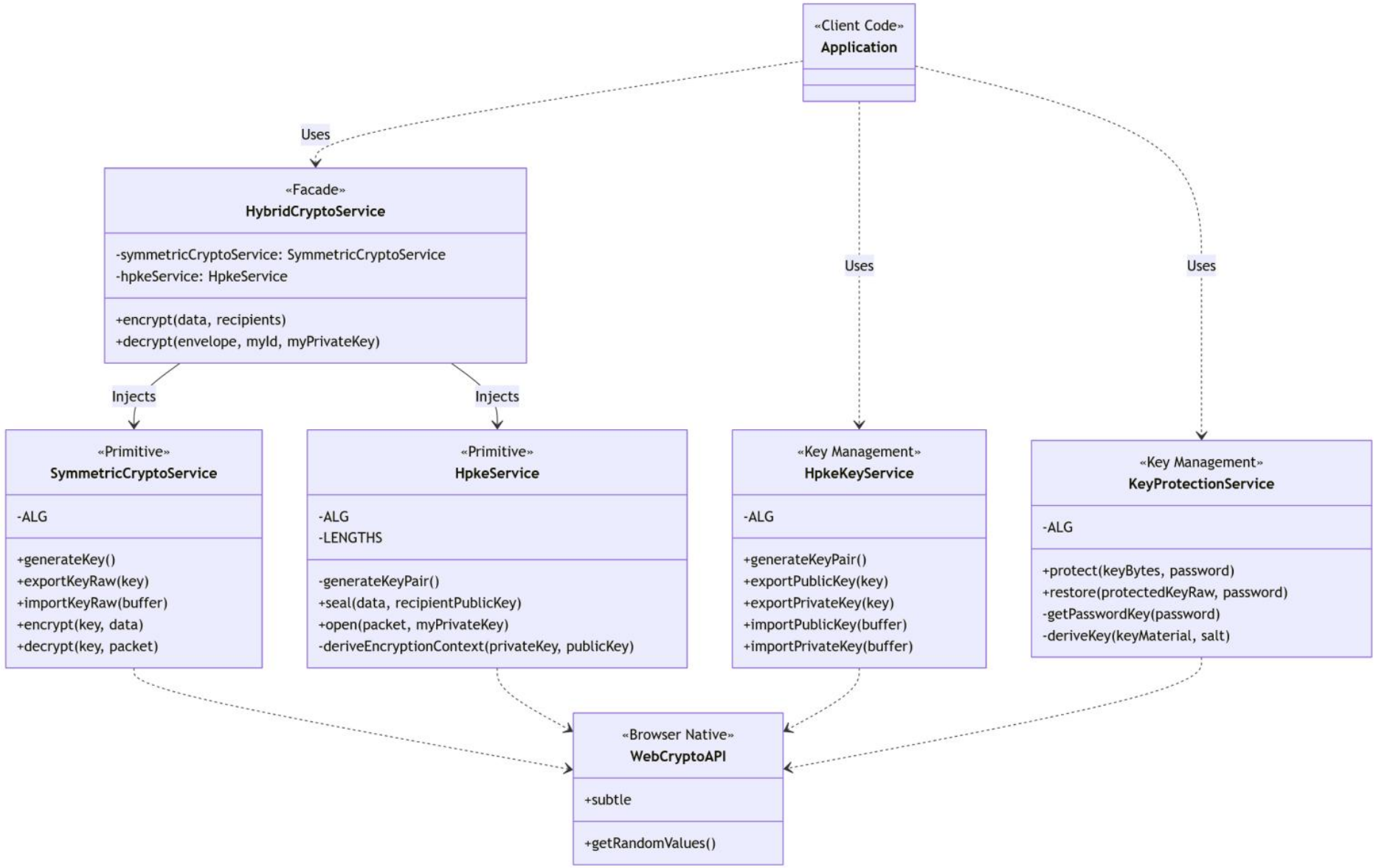
Реалізація криптографічного модуля є найбільш критичною частиною системи, оскільки будь-яка помилка в логіці або неправильне використання примітивів може призвести до компрометації даних. Тому при розробці цього модуля головний акцент було зроблено на суворій типізації, ізоляції відповідальності та передбачуваності поведінки.

Для написання модуля обрано мову TypeScript. Використання статичної типізації дозволяє на етапі компіляції виключити цілий клас помилок, пов’язаних з некоректною обробкою типів даних.

Архітектура ядра базується на об’єктно-орієнтованому підході з використанням принципів SOLID. Для організації коду та управління залежностями застосовано наступні патерни проектування: впровадження залежностей, одинак, фасад.



Діаграма класів ядра шифрування



Загальні висновки та результати

У роботі спроектовано архітектуру веб-системи захищеного обміну повідомленнями в реальному часі на принципах Zero-Knowledge. Обгрунтовано та реалізовано гібридну архітектуру наскрізного шифрування на базі стандартів HPKE та AES-GCM з використанням Web Cryptography API. Вирішено задачі безпечного управління ключами, синхронізації сесій та потокового шифрування великих масивів даних у веб-середовищі.

Запропоновані механізми шифрування та безпечного управління ключами, які можуть слугувати основою для побудови захищених застосунків або інтегровані в наявні, що потребують підвищення рівня захисту даних.

Дякую за увагу!