Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

«Допущено до захисту»
В.о. завідувача кафедри БІСТ Мелкозьорова О.М.

_____
« » 2024 р.

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Розробка та дослідження програмних методів протидії спотворенню даних у системах зберігання хмарних обчислень»

Оцінка « »

Голова ЕК

Лемешко О.В. _____

Керівник : к. т. н. Громико І.О.

Рецензент: д. т. н. Краснобаєв В.А.

Виконавець : студент групи КБ-42

_____ Яшин А.П.

Харків 2024

РЕФЕРАТ

Дипломна робота включає 58 сторінок, 21 рисунок, 1 додаток, і список літератури з 21 джерела.

Мета роботи полягає у створенні програмного рішення, здатного захищати дані від несанкціонованого доступу та забезпечувати їх цілісність через шифрування та цифрові підписи. Використані методи дослідження включають аналітичний огляд сучасних кіберзагроз, розробка програмного рішення та його тестування з використанням криптографічних бібліотек та інструментів захисту даних.

Основним результатом стало розроблення програмного рішення та проведення його тестування з використанням криптографічних бібліотек та інструментів захисту даних, на підставі дослідження програмних методів протидії спотворенню даних у системах зберігання хмарних обчислень, з точки зору аналітичного огляду сучасних кіберзагроз. Дані методи можуть бути використані у банківській сфері, системах здоров'я та інших областях, де важливий захист інформації.

Рекомендації щодо використання включають інтеграцію розробленого рішення в існуючі хмарні платформи. Значущість роботи підкреслюється її актуальністю у контексті зростаючих загроз кібербезпеці. Можливі напрямки подальших досліджень включають розширення функціональності системи, зокрема, розробка механізмів штучного інтелекту для прогнозування та запобігання кібератак.

Ключові слова: ХМАРНІ ОБЧИСЛЕННЯ, ШИФРУВАННЯ ДАНИХ, ЦИФРОВИЙ ПІДПИС, КІБЕРБЕЗПЕКА, КРИПТОГРАФІЯ, ЗАХИСТ ІНФОРМАЦІЇ, АУТЕНТИФІКАЦІЯ.

ABSTRACT

Volume of work: 58 pages, 21 figures, 1 appendix, and a bibliography with 21 sources.

The aim of the work is to develop a software solution capable of protecting data from unauthorized access and ensuring its integrity through encryption and digital signatures. The research methods used include an analytical review of contemporary cyber threats, the development of the software solution, and its testing using cryptographic libraries and data protection tools.

The main outcome of this project was the development and testing of the software solution utilizing cryptographic libraries and data protection tools, based on the study of software methods to counteract data corruption in cloud storage systems, from the perspective of an analytical review of modern cyber threats. These methods can be employed in the banking sector, health systems, and other areas where information protection is critical.

Recommendations for use include the integration of the developed solution into existing cloud platforms. The significance of the work is underlined by its relevance in the context of increasing cybersecurity threats. Potential directions for further research include the expansion of the system’s functionality, particularly the development of artificial intelligence mechanisms for predicting and preventing cyberattacks.

Keywords: CLOUD COMPUTING, DATA ENCRYPTION, DIGITAL SIGNATURE, CYBERSECURITY, CRYPTOGRAPHY, INFORMATION PROTECTION, AUTHENTICATION.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AES – Advanced Encryption Standard – Розширений стандарт шифрування

DES – Data Encryption Standard – Стандарт шифрування даних

ECC – Elliptic Curve Cryptography – Криптографія на еліптичних кривих

RSA – Rivest–Shamir–Adleman (algorithm) – Алгоритм Рівеста–Шаміра–Адлемана

CDN – Content Delivery Network – Мережа доставки контенту

SLA – Service Level Agreement – Угода про рівень обслуговування

GDPR – General Data Protection Regulation – Загальне регулювання захисту даних

API – Application Programming Interface – Інтерфейс програмування додатків

DoS – Denial of Service – Відмова у обслуговуванні

CRM – Customer Relationship Management – Управління відносинами з клієнтами

IoT – Internet of Things - Інтернет речей

SHA – Secure Hash Algorithm – Алгоритм криптографічного хешування

ШІ – Штучний інтелект

ВСТУП

У зв'язку зі стрімким розвитком інформаційних технологій та швидким приростом обсягів даних, виникає необхідність у забезпеченні безпеки, цілісності та конфіденційності цих даних. Хмарні обчислення стали основою для зберігання та обробки великих обсягів інформації, але одночасно постали нові виклики, пов'язані з безпекою цих даних. Проблема спотворення та пошкодження даних стала серйозною загрозою, що вимагає розробки та впровадження ефективних методів захисту.

Історія захисту даних сягає своїм корінням в давнину, коли для захисту важливої інформації використовувались шифри та коди. З розвитком технологій і збільшенням обсягів обробки даних, постали нові методи та технології захисту. Наприклад, виникли методи шифрування даних, які дозволяли захищати інформацію від несанкціонованого доступу.

У більш сучасному контексті, з появою хмарних обчислень, проблема забезпечення безпеки даних стала більш актуальною. З одного боку, хмарні обчислення надають більше можливостей для зберігання та обробки даних, а з іншого - вони вносять нові виклики, пов'язані з безпекою і конфіденційністю цих даних.

На сьогоднішній день, з розвитком кіберзлочинності та зростанням кількості інформації, що обробляється у хмарних сервісах, проблема спотворення даних стала більш актуальною та складною. Захист інформації стає завданням номер один для організацій та користувачів, які використовують хмарні сервіси для зберігання та обробки даних.

У зв'язку з цим, розробка програмних методів протидії спотворенню даних набуває великого значення. Ці методи мають забезпечити не лише безпеку даних, але й їх цілісність та конфіденційність у хмарних обчисленнях.

Інформаційна революція та стрімкий розвиток технологій призвели до вибухового зростання обсягів даних, що зберігаються та оброблюються у

хмарних середовищах. Цей надмір даних створює складнощі у забезпеченні їх безпеки та цілісності.

«Із збільшенням кількості даних у хмарних системах, зростає також і ризик кібератак та спотворення інформації» [5]. Зловмисники постійно шукають нові способи отримання несанкціонованого доступу до цих даних, використовуючи різноманітні атаки та техніки.

Організації та користувачі вкладають значні зусилля в забезпечення безпеки своїх даних, оскільки їхнє виток або спотворення може призвести до серйозних наслідків, таких як втрата конфіденційності, фінансові збитки або порушення репутації.

Хмарні обчислення мають свої особливості, зокрема розподіленість та віртуалізація, які створюють нові виклики у забезпеченні безпеки даних. Відсутність прямого контролю над інфраструктурою може ускладнити виявлення та вирішення проблем безпеки.

Розробка та дослідження програмних методів протидії спотворенню даних є актуальним завданням у сучасному світі хмарних обчислень. Ці методи повинні забезпечувати надійний захист даних у хмарних сервісах, зберігаючи їх цілісність та конфіденційність.

Програмні методи відіграють важливу роль у забезпеченні безпеки даних у хмарних обчисленнях. Вони надають можливість розробляти та застосовувати ефективні заходи захисту, які допомагають у запобіганні та виявленні спотворення даних. Програмні методи шифрування даних використовуються для захисту їх від несанкціонованого доступу під час транспортування та зберігання. Застосування сучасних алгоритмів шифрування забезпечує високий рівень конфіденційності даних, навіть у випадку їх незаконного доступу. «Важливість розробки програмних методів для захисту даних у хмарних сервісах підкреслюється потребою у високому рівні захисту для запобігання спотворенню інформації, яке може призвести до значних втрат для компаній та користувачів» [6].

Використання цифрових підписів дозволяє перевіряти цілісність та автентичність даних. Програмні методи цифрового підпису забезпечують можливість підписувати дані та перевіряти їх підписи для виявлення будь-яких змін або спотворень.

Хеш-функції використовуються для створення унікальних "відбитків" даних, що дозволяє виявляти навіть малі зміни в них. Програмні методи хешування можуть бути застосовані для перевірки цілісності даних під час їх передачі або зберігання.

Програмні методи аутентифікації та авторизації дозволяють перевіряти права доступу користувачів до даних у хмарних обчисленнях. Вони забезпечують контроль над тим, хто має доступ до яких даних, що допомагає у запобіганні несанкціонованого доступу та зловживань. Ці методи забезпечують не тільки захист даних від несанкціонованого доступу, але й гарантують їх цілісність під час передачі та зберігання в хмарних системах [8]. Програмні методи для моніторингу та аудиту безпеки дозволяють виявляти та реагувати на можливі загрози та атаки на дані у реальному часі. Вони надають можливість відстежувати активність користувачів та систем, а також реагувати на підозрілі події. Сучасні програмні методи використовують штучний інтелект та машинне навчання для виявлення та прогнозування потенційних загроз для даних у хмарних обчисленнях. Вони аналізують великі обсяги даних, щоб виявляти аномалії та патерни, що вказують на можливі атаки.

Програмні методи протидії спотворенню даних у хмарних обчисленнях є важливими для забезпечення надійності, конфіденційності та цілісності інформації у сучасному цифровому середовищі. Вони допомагають організаціям та користувачам зберігати довіру до хмарних сервісів та ефективно захищати їх дані від різноманітних загроз.

Мета роботи полягає в розробці та дослідженні програмних методів протидії спотворенню даних у системах зберігання хмарних обчислень. Основні завдання включають:

- Аналіз сучасних проблем та викликів, пов'язаних з безпекою даних у хмарних обчисленнях.
- Розробка програмних методів захисту даних від спотворення та несанкціонованого доступу.
- Використання різноманітних криптографічних методів, включаючи шифрування, цифрові підписи та хешування.
- Реалізація та тестування розроблених програмних методів на практиці.
- Оцінка ефективності та надійності застосованих методів у контексті захисту даних у хмарних обчисленнях.

Галузь застосування результатів охоплює широкий спектр областей, де використовуються хмарні обчислення, включаючи бізнес, науку, медицину, фінанси та інші. Результати дослідження можуть бути корисними для організацій, що мають справу з обробкою та зберіганням великих обсягів конфіденційної інформації у хмарних сервісах. Вони допоможуть забезпечити високий та надійний рівень захисту даних та не покладатися лише на довіру до хмарних обчислень.

1 ХМАРНІ ОБЧИСЛЕННЯ

1.1 Що таке хмарні обчислення.

Хмарні обчислення – це доставка IT-ресурсів через Інтернет на вимогу з оплатою за фактом використання. Тобто орендувати будь-які ресурси, наприклад, сервери (не купувати). Крім серверів, є купа інших речей, які поставляються з хмарними сервісами. Хмарні обчислення визначаються як доставка обчислювальних ресурсів через Інтернет, де користувачі можуть отримувати доступ та управляти ресурсами без прямого активного управління фізичними серверами [1]. Тепер давайте обговоримо, що таке хмарні обчислення на прикладі серверів – найпростіших для розуміння та пояснення.

Скажімо, у нас є унікальна інтернет-платформа, де люди можуть купувати, продавати, обмінювати різні книги. Припустимо, що ідея вже розроблена і є код і всі необхідні розрахунки і план цього проекту. Що залишається? Підняти цей додаток в інтернеті - значить запустити його в продакшн, щоб користувачі з усього світу могли використовувати цю платформу.

Тепер подивимося, як ця ідея буде реалізована в традиційному варіанті – без використання хмарних технологій. Що нам для цього знадобиться:

- Фізичні сервери для баз даних, Інтернету і т.д., де ми будемо встановлювати наше програмне забезпечення.
- Фізичне місце розташування цих серверів – приміщення типу дата-центру. Орендуйте приміщення або купіть приміщення, позначте там всі наші фізичні сервери, підключіть все, налаштуйте.
- Необхідно подбати про охолодження в цьому приміщенні. Для того, щоб нічого не перегрівалося, необхідно забезпечити кондиціонування цього центру.
- Доступ до мережі Інтернет, причому на високій швидкості.
- Звичайно ж, електрика.

- Фізична охорона. Люди, які будуть перешкоджати несанкціонованому проникненню в приміщення.
- Працівники для підтримки серверів. У разі несправності сервера потрібні фахівці для підтримки роботи центру.

Ці предмети спричиняють не тільки матеріальні втрати, а й чималу кількість часу на організацію та налаштування, яке може тривати місяцями.

А тепер уявімо, що ідея стала чудовою і користувачів з різних куточків світу стає все більше. Додаток починає зависати і давати збої. Приходить логічне рішення створити такий же центр в абсолютно різній частині світу. І з чим ми зіткнулися? З усіма перерахованими вище проблемами і витратами. А на цей раз вам знадобиться ще більше часу, так як регіон вам незнайомий.

І в разі невдачі приміщення, сервери, люди залишаються непотрібними і просто марною тратою часу і грошей. Що пропонує хмарний сервіс?

Що вимагається від нас як від споживача? Будь-який комп'ютер, ноутбук з браузером. І інтернет, але його швидкість не важлива, навіть якщо це публічний інтернет на вулиці з якогось кафе. А що з термінами. Він необхідний тільки для настройки. Ніякого клопоту з фізичними серверами, людьми, кабелями, електрикою та приміщенням, все це зайве. Все робиться за лічені секунди, хвилини.

Незаперечною перевагою є те, що не потрібно гадати, чи буде проект успішним чи ні, скільки серверів знадобиться в даний момент. На рисунку 1.1 показано графік відповідності існуючої потужності до навантаження програми використовуючи традиційні варіанти.

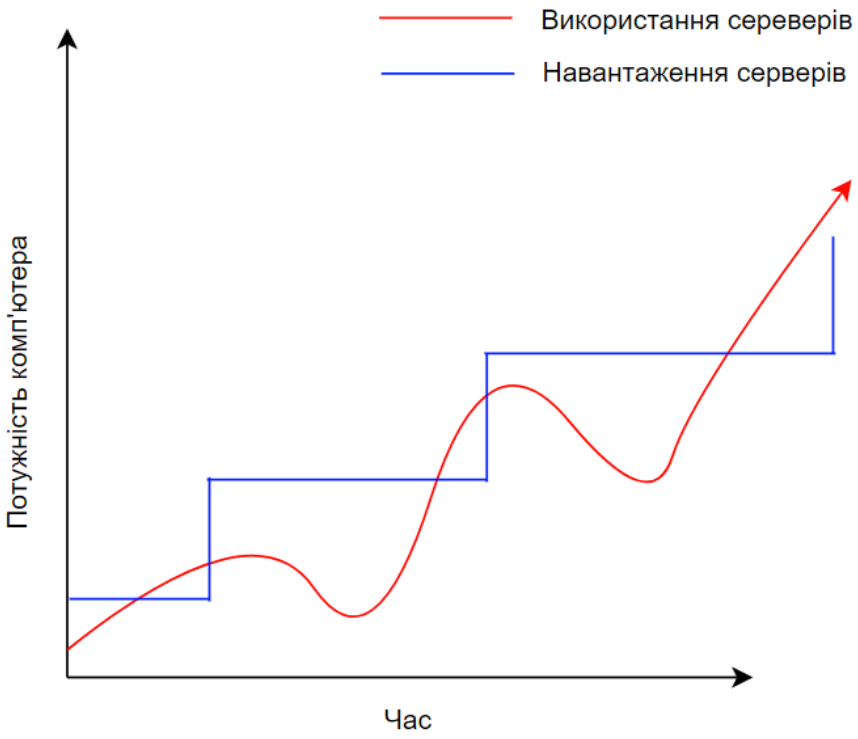


Рисунок 1.1 – Графік відповідності існуючої потужності до навантаження програми у традиційному варіанті

Припустимо, ми спочатку купуємо 20 серверів і отримуємо відповідну потужність (синя лінія на графіку). Червоний рядок показує використання наших серверів, тобто фактичне навантаження програми. Бізнес потихеньку йде вгору і потужностей не вистачає. У цьому випадку потрібно купити певну кількість серверів. Однак раптом, несподівано, відбувається стрибок не на нашу користь і користувачів дуже мало. При цьому більшість наших серверів є незатребуваними. Гірше, якщо кількість користувачів буде стрімко збільшуватися, а сервери не зможуть впоратися з напливом відвідувачів і виникне маса збоїв і проблем, які природним чином позначаться на популярності продукту і невдоволенні клієнтів, як показано на графіку. Така ситуація призведе до величезної втрати репутації і, звичайно ж, грошей. (Графіки було побудовано за допомогою Draw.io [21]). На рисунку 1.2 можна побачити ту саму відповідність, але використовуючи хмарні технології.

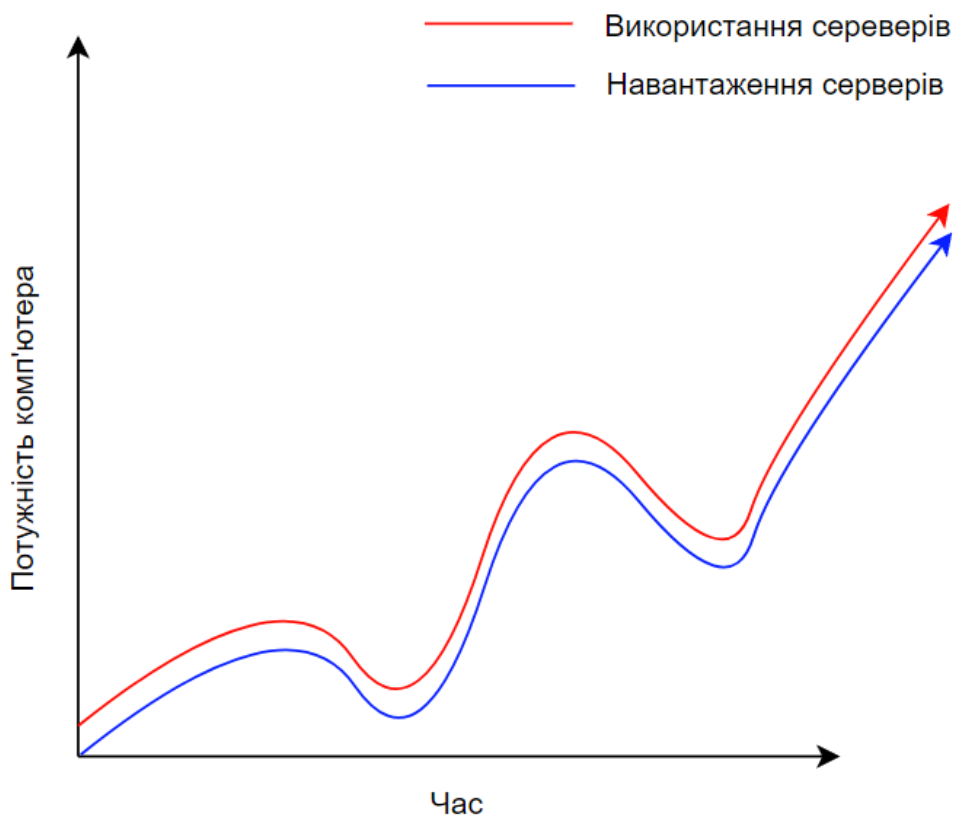


Рисунок 1.2 – Графік відповідності існуючої потужності до навантаження програми у разі використання хмарних технологій

Що ми отримуємо, переходячи на хмарну інфраструктуру? Як видно з графіка, забезпечується еластичність, тобто ми отримуємо необхідні нам ресурси за лічені секунди і не втрачаємо таким чином продуктивність. Крім того, є можливість зменшити кількість використовуваних ресурсів у разі низької активності (наприклад, у нічний час) для економії коштів. При цьому оплата здійснюється за ресурси, які ми реально використовуємо, і нічого зайвого. Робиться це просто, при налаштуванні ресурсів. Одним із популярних варіантів є те, що якщо завантаження ЦП перевищує 80% за останні 15 хвилин, додайте один веб-сервер, якщо завантаження ЦП менше 20% за ті ж 15 хвилин, видаліть один сервер відповідно. Ось у чому різниця – хмара надає ресурсам еластичності.

1.2 Історія хмарних обчислень.

Для того, зоб глибше розглянути представлену тему, розглянемо історію хмарних обчислень. Також охопимо історію клієнт-серверних обчислень, розподілених обчислень і хмарних обчислень.

До появи обчислень використовувалася клієнт-серверна архітектура, в якій всі дані та управління клієнтом знаходяться на стороні сервера. Якщо один користувач хоче отримати доступ до деяких даних, спочатку він мусить підключитися до сервера, а потім отримає відповідний доступ. Але в цьому було багато недоліків.

Розглянемо деякі з них.

- Централізація ресурсів та управління. Всі дані та управління процесами централізовано на сервері, що створює одну точку відмови. Якщо сервер падає, всі клієнти, пов'язані з ним, втрачають доступ до ресурсів та послуг. Якщо сервер бази даних зазнає збою, всі користувачі втрачають доступ до важливих даних, що може зупинити виробничий процес у компанії.
- Масштабування. Масштабування клієнт-серверних систем часто вимагає додаткового апаратного забезпечення або збільшення потужності сервера, що може бути дорогим, нудним та складним. Додавання нових працівників до компанії може вимагати встановлення додаткових серверів для збереження та обробки збільшеного об'єму даних.
- Доступність. У клієнт-серверних архітектурах ресурси сервера мають обмежену доступність, особливо при великому навантаженні або проблемах з мережею. В годину пік, коли всі працівники намагаються доступитися до корпоративної CRM системи, сервер може перевантажуватися, викликаючи сповільнення або недоступність сервісу. Безпека. Оскільки всі дані зосереджені в одному місці, системи стають більш уразливими для атак, які можуть призвести до втрати або витоку даних. Хакерська атака на сервер може призвести до втрати конфіденційної інформації клієнтів.
- Продуктивність. Якість обслуговування може погіршуватися зі збільшенням кількості користувачів, оскільки всі запити обробляються

центральним сервером. Інтернет-магазин під час, наприклад, чорної п'ятниці може відчувати сповільнення через збільшене навантаження на сервер.

- Затримки в мережі. У клієнт-серверних архітектурах, особливо в географічно розподілених системах, затримки в мережі можуть впливати на продуктивність. Користувачі з різних країн, які спробують доступитися до централізованого сервера, можуть зіштовхнутися з високою затримкою, що погіршує їхній досвід користування.
- Витрати на обслуговування. Великі клієнт-серверні системи вимагають значних витрат на обслуговування, включаючи апаратне обслуговування, оновлення програмного забезпечення та забезпечення безперебійної роботи. Велика компанія може витрачати значні суми на забезпечення обслуговування своїх серверів, що включає резервне копіювання даних, оновлення безпеки та ремонт апаратури.

Ці недоліки спонукали розвиток нових технологій та парадигм, зокрема хмарних обчислень, які дозволили розподілити обчислювальні та зберігаючі ресурси, підвищити масштабованість, доступність, а також знизити витрати на обслуговування. Тому, після клієнт-серверних обчислень, з'явилися розподілені обчислення, в яких всі комп'ютери були з'єднані мережею, що дозволяло користувачам ділитися своїми ресурсами, коли це було необхідно. Вона також мала певні обмеження та виклики. Таким чином, щоб позбутися обмежень розподіленої системи, було розроблено хмарні обчислення.

У 1961 році John McCarthy у своїй промові в MTI сказав, що "обчислення можна купувати як комунальну послугу, як воду та електрику" [2]. За словами інформатика, це була чудова ідея. Але люди того часу не хотіли приймати цю технологію. Вони думали, що технологія, яку вони використовують, достатньо ефективна для них. Таким чином, ця концепція обчислень не була особливо оцінена, і дуже мало досліджень було проведено в цьому напрямку. Але з часом технологія ухопила ідею, і через кілька років ця ідея була реалізована. Так, це було реалізовано в 1999 році компанією Salesforce.com.

Ця компанія почала надавати підприємницькі додатки через Інтернет, і таким чином почався розвиток хмарних обчислень.

У 2002 році Amazon започаткував Amazon Web Services (AWS), Amazon надаватиме сховища, обчислення через Інтернет. У 2006 році Amazon запустив Elastic Compute Cloud Commercial Service, який був відкритий для всіх, став революційним у наданні широкого спектру хмарних обчислювальних ресурсів [3].

Після цього у 2009 році Google Play також почав надавати хмарні обчислення для підприємств, як інші компанії, які побачили появу хмарних обчислень, також почали надавати свої хмарні сервіси. Так, у 2009 році Microsoft запустив Microsoft Azure, а потім інші компанії, такі як Alibaba, IBM, Oracle, HP, також представили свої хмарні послуги. У наші дні хмарні обчислення стають дуже популярними та важливими навичками.

1.3 Переваги та недоліки хмарних обчислень

Переваги хмарних обчислень включають гнучкість та швидкість розгортання ресурсів, економію витрат за рахунок плати лише за використані ресурси, масштабованість та доступність ресурсів з будь-якої точки світу [4]. Як і майже все а світі хмарна інфраструктура пропонує ряд переваг, які зробили її популярним рішенням для бізнесів усіх розмірів, та недоліків, що вносить ложку дьогтю у цей мед.

Розглянемо детальніше деякі з переваг:

- Гнучкість: від ідеї до реалізації за лічені хвилини. Хмарні платформи надають необмежену гнучкість у розгортанні та управлінні IT-ресурсами. Компанії можуть швидко розробляти та запускати нові додатки, оскільки не потрібно інвестувати в апаратне забезпечення або чекати його розгортання. Це дає можливість реагувати на зміни ринкових умов або бізнес-вимог з небувалою швидкістю.
- Еластичність: автоматично розподіляйте ресурси, які ви фактично використовуєте. Хмарна інфраструктура дозволяє масштабуватися згідно з потребами. Ресурси (як-от обчислювальна потужність, пам'ять і сховище)

можуть бути автоматично збільшені або зменшені в залежності від навантаження, забезпечуючи оптимальну продуктивність і ефективність використання.

- Економія витрат: змінні витрати за рахунок масштабування. Використання хмарних послуг знижує загальні витрати на IT, оскільки компанії платять тільки за ті ресурси, які вони використовують - зручно. Це відрізняється від традиційного підходу з необхідністю робити значні капітальні інвестиції в обладнання, яке може бути не завжди використане на повну потужність.
- Глобальний: мультирегіональна інфраструктура за лічені хвилини. Хмарні постачальники пропонують глобальну інфраструктуру, яка дозволяє компаніям розгортати служби в мультирегіональних дата-центрах. Це забезпечує високу доступність та кращу продуктивність для користувачів у різних частинах світу, оптимізуючи локалізацію контенту і скорочуючи затримку.

Ці переваги роблять хмарні обчислення ідеальним рішенням для компаній, які прагнуть до інновацій, шукають шляхи зниження витрат, прагнуть до глобального розширення або потребують високої гнучкості у своїх IT-операціях. Крім того, екологічні переваги, такі як зменшення вуглецевого сліду завдяки більш ефективному використанню ресурсів і зменшення необхідності фізичних дата-центрів, роблять хмарні рішення ще привабливішими навіть для критиків.

Хмарні обчислення пропонують численні переваги, але як і будь-яка технологія, вони мають свої недоліки. Ось декілька потенційних недоліків хмарної інфраструктури:

- Залежність від Інтернету. Хмарні сервіси вимагають стабільного та швидкісного Інтернет-з'єднання. Перебої з інтернетом можуть призвести до втрати доступу до важливих даних та додатків, що може серйозно вплинути на бізнес-операції.

- Проблеми конфіденційності та безпеки. Хмарні сервіси зберігають дані на серверах, які фізично розташовані поза контролем користувача, що може підвищити ризик витоку або неналежного використання даних. Хоча провайдери хмарних сервісів зазвичай застосовують суворі заходи безпеки, ризик втрати даних або хакерських атак все ж існує.
- Обмежена контроль та гнучкість. Користувачі хмарних сервісів часто мають менше контролю над своїми інфраструктурними ресурсами порівняно з традиційними внутрішніми розгортаннями. Це може включати обмеження в налаштуваннях конфігурацій, програмного забезпечення та інших аспектів системи.
- Вартість. Хоча хмарні обчислення можуть знижувати витрати на IT у деяких випадках, вони також можуть бути дорогими у довгостроковій перспективі, особливо якщо користувачі постійно збільшують кількість використовуваних ресурсів. Моделі оплати за використання можуть швидко збільшити витрати, особливо при високому використанні даних або обчислювальної потужності.
- Залежність від постачальника. Компанії можуть стати залежними від одного хмарного постачальника (vendor lock-in), що ускладнює перехід на інші платформи або повернення до внутрішньої інфраструктури без значних затрат часу та ресурсів.
- Юридичні та договірні питання. Управління даними в хмарі може підняти питання щодо відповідності законодавчим та нормативним вимогам, особливо коли дані зберігаються у міжнародних дата-центрах. Компанії мають бути уважними щодо того, як і де їхні дані зберігаються та обробляються для відповідності GDPR та іншим нормативним вимогам.

Ці недоліки вимагають ретельного планування та управління ризиками, коли компанії обирають хмарні рішення для своїх потреб. Важливо обирати надійних постачальників хмарних послуг і мати чітке розуміння угод про рівень сервісу (SLA) та політик безпеки, що застосовуються.

SLA (Service Level Agreement) або угода про рівень обслуговування — це офіційний документ або контракт між постачальником послуг та його клієнтом, який визначає обсяг наданих послуг і стандарти, за якими ці послуги повинні бути надані. Угода SLA містить детальні параметри і очікування, щоб забезпечити обидві сторони чітким розумінням стандартів обслуговування. SLA має критичне значення, оскільки воно допомагає забезпечити, що обидві сторони мають чітке розуміння очікувань та вимог, що сприяє більш ефективній і злагодженій роботі. Угоди SLA також допомагають уникнути непорозумінь і забезпечують механізм вирішення спорів, що може заощадити час та ресурси обох сторін. Вони є фундаментальною частиною управління відносинами з клієнтами в IT-секторі та інших областях, де якість і надійність послуг є важливими.

1.4 Витоки інформації з хмарних середовищ

Витоки інформації з хмарних середовищ можуть призводити до серйозних наслідків, зокрема до втрати даних, порушення конфіденційності та фінансових збитків для компаній та індивідуальних користувачів. Розглянемо декілька прикладів значних витоків даних з хмарних середовищ, які стались у останні роки:

- Capital One (2019) Capital One, одна з найбільших банківських корпорацій США, стала жертвою витоку даних, у результаті якого особиста інформація близько 106 мільйонів людей була викрадена. Інцидент трапився через неналежне конфігурування веб-додаткового фаєрвола в середовищі Amazon Web Services, яке використовувалося банком [19].
- Verizon (2017) Близько 6 мільйонів записів клієнтів Verizon були витечені через помилкову настройку хмарного сховища Amazon S3, яке налаштував один із партнерів Verizon. Ці записи містили інформацію про клієнтів, включно з їх телефонними номерами [11].
- Adobe Creative Cloud (2013) Adobe оголосила про витік даних, що стосувався приблизно 38 мільйонів користувачів. Цей витік включав імена користувачів, зашифровані паролі, а також дані кредитних карт. Хоча Adobe

широко використовує хмарні сервіси для зберігання даних, цей інцидент підкреслив важливість впровадження додаткових заходів безпеки [9].

- Dropbox (2012) Dropbox зіткнувся з витоком імен користувачів та паролів, що були використані для несанкціонованого доступу до близько 68 мільйонів облікових записів. Витік стався через застарілі паролі користувачів та відсутність двофакторної аутентифікації в принципі на той час [10].

Ці інциденти підкреслюють важливість впровадження комплексних заходів безпеки у хмарних обчисленнях. Результати дослідження та розробки програмних методів протидії спотворенню даних, які розглядаються у цій роботі, стають ключовим елементом в забезпеченні безпеки та цілісності інформації. Впровадження розроблених програмних методів дозволить зменшити ризики пов'язані з використанням хмарних сервісів та мінімізувати можливість спотворення даних. Крім того, регулярний аудит системи, включаючи перевірку ефективності розроблених програмних заходів, є невід'ємною частиною процесу забезпечення безпеки даних у хмарних обчисленнях. Отримані результати дослідження можуть служити основою для покращення та доповнення існуючих заходів безпеки, спрямованих на запобігання інцидентам спотворення даних у хмарних обчисленнях.

2 ЗАХИСТ ІНФОРМАЦІЇ У ХМАРНИХ СЕРВІСАХ ЧЕРЕЗ ШИФРУВАННЯ ТА ЦИФРОВІ ПІДПИСИ

Хмарні обчислення трансформували способи зберігання, обробки та управління даними в сучасному бізнес-світі. Однак, зростання залежності від хмарних технологій також збільшило ризики пов’язані з безпекою даних. Шифрування у хмарному середовищі є критично важливим для забезпечення конфіденційності, інтеграції та доступності даних, а також для відповідності нормативним стандартам.

Конфіденційність даних у хмарному середовищі є однією з основних турбот користувачів і компаній, які вирішують перейти на хмарні рішення. Ключова мета конфіденційності даних полягає у забезпеченні того, що інформація доступна тільки для тих, хто має на це право. Загрози можна поділити на внутрішні та зовнішні [12].

Внутрішні загрози означають дії, що виникають всередині організації, і часто пов'язані зі співробітниками, підрядниками, або будь-якою особою, яка має легітимний доступ до системи. Ці загрози можуть бути навмисними або випадковими. Розглянемо деякі з таких атак.

- Інсайдерські атаки. Співробітники або колишні співробітники використовують свої знання та доступ для вчинення шкідливих дій, таких як крадіжка, видалення, або зміна даних.
- Помилкові дії. Випадкові дії, такі як неправильне налаштування системи, введення помилкових даних, або випадкове видалення важливих файлів, що можуть спричинити значні збої в роботі системи або взагалі «покласти» сервіс.
- Недотримання процедур безпеки. Відсутність дотримання встановлених правил безпеки, недбалість у виконанні обов'язків, що призводить до вразливостей в системі.

Зовнішні загрози виникають із зовні і зазвичай включають дії, спрямовані на отримання несанкціонованого доступу до системи.

- Хакерські атаки: Включають широкий спектр атак, таких як "відмова в обслуговуванні" (DoS), фішинг, встановлення шкідливого програмного забезпечення, а також більш складні кібератаки з використанням вразливостей у програмному забезпеченні.
- Розповсюдження вірусів та шкідливих програм. Цілеспрямоване введення шкідливих програм у систему з метою викрадення, зміни або знищення даних.
- Атаки на фізичну інфраструктуру. Спроби фізичного доступу до центрів обробки даних або інших критичних елементів інфраструктури для виконання шкідливих дій або знищення обладнання. Аналіз вразливостей системи, що може сприяти таким загрозам.

Диму без вогню не буває, тому розглянемо вразливості, які можуть слугувати подразником для атак.

- Технічні вразливості. Недоліки у конфігурації безпеки: Недостатньо сильні паролі, відсутність мультифакторної аутентифікації, та неадекватні політики доступу можуть спричинити непомічені проникнення в систему.
- Застаріле програмне забезпечення. Використання програмного забезпечення без останніх оновлень безпеки залишає систему вразливою до відомих атак.
- Недостатнє шифрування. Використання слабких алгоритмів шифрування або відсутність шифрування чутливих даних може дозволити зловмисникам легко отримати доступ до цих даних.
- Вразливості програмного забезпечення. Баги та недоліки в програмному коді, які можуть бути використані для несанкціонованого доступу або виконання шкідливих скриптів.

Не слід виключати з уваги організаційні вразливості:

- Відсутність програми навчання з безпеки. Співробітники, які не обізнані з кращими практиками та принципами кібербезпеки, можуть несвідомо створювати вразливості.

- Слабке управління ідентифікацією та доступом. Неналежне управління ідентифікацією та контролем доступу до ресурсів може дозволити несанкціонований доступ до критично важливих систем.
- Відсутність регулярного аудиту безпеки. Регулярний аудит безпеки допомагає виявляти та виправляти уразливості перед тим, як їх може використати зловмисник.

Також не менш важливими є процедурні вразливості.

- Неконтрольовані зміни в системі: Відсутність процедур контролю за змінами може призвести до введення непомічених уразливостей.
- Неналежне розслідування інцидентів безпеки: Відсутність адекватних процедур для розслідування та реагування на інциденти безпеки може призвести до повторення подібних інцидентів.

Захист хмарних обчислень вимагає комплексного підходу до кібербезпеки, що включає технічні, організаційні та процедурні аспекти, а також постійну оцінку та покращення захисних механізмів для адаптації до нових загроз та вразливостей [13].

Всі перераховані вразливості та проблеми потребують рішення. Звичайно, всі проблеми одним додатком не вирішити, але навіть спроба усунути деякі з них буде корисною. Хоча, ми, як звичайні користувачі, не маємо змоги фактично вплинути на процедуру забезпечення конфіденційності й цілісності наших даних, в наших силах попередньо, превентивно зробити так, щоб навіть у разі витоку нашої інформації, вона мала ще декілька шарів безпеки. Шифрування точно не буде зайвим, та зробить дані не такою привабливою ціллю для зловмисників.

Шифрування є одним з найефективніших способів захисту даних в хмарному середовищі. Воно допомагає вирішити багато з проблем, описаних вище, забезпечуючи конфіденційність, цілісність та доступність даних. Ось як шифрування може вплинути на основні аспекти безпеки хмарних обчислень, пробіжимося по кожному з пунктів:

- Захист конфіденційності: Шифрування перетворює чутливі дані в зашифрований формат, який можна прочитати тільки за допомогою відповідного ключа. Це запобігає несанкціонованому доступу до інформації, навіть якщо дані викрадені або ненавмисно витекли.
- Захист від зовнішніх та внутрішніх загроз: Зашифровані дані залишаються безпечними навіть у випадку злому хмарної інфраструктури, знижуючи ризики, пов'язані з кібератаками або несанкціонованими діями співробітників.
- Відповідність законодавчим вимогам: Багато нормативних актів вимагають, щоб чутливі дані були належним чином захищені. Шифрування допомагає організаціям дотримуватися таких вимог, забезпечуючи високий рівень захисту даних.
- Захист від помилок і атак: Шифрування знижує ризики, пов'язані з помилковими налаштуваннями безпеки або вразливостями системи, які можуть бути використані зловмисниками для отримання доступу до даних.

Використання цих методів шифрування, в залежності від вимог до безпеки, може значно підвищити безпеку хмарних даних та захистити організації від потенційних загроз та атак.

Шифрування на стороні клієнта. Перш ніж дані відправляються у хмару або вводяться в хмарну інфраструктуру, їх можна зашифрувати на стороні клієнта, що гарантує, що вони вже зашифровані при передачі та зберіганні.

Шифрування на стороні сервера. Хмарні провайдери також застосовують шифрування для захисту даних, збережених на їхніх серверах. Це забезпечує додатковий рівень безпеки, навіть якщо периметр фізичної безпеки було порушено.

Ці заходи забезпечують конфіденційність даних у хмарі і допомагають запобігти їх несанкціонованому використанню чи витоку.

Зовнішні та внутрішні загрози в хмарних обчисленнях можуть мати серйозні наслідки для безпеки даних. Шифрування є одним із способів мінімізації ризиків, пов’язаних з цими загрозами.

Розглянемо зовнішні загрози. Це можуть бути хакерські атаки, фішинг, програми-вимагачі та інші маліційні дії, спрямовані на отримання несанкціонованого доступу до хмарних даних. Шифрування знижує цінність перехоплених даних для зловмисників, оскільки без ключа дешифрування дані залишаються недоступними.

Також важливо пам’ятати і про внутрішні загрози. Це можуть бути співробітники провайдера хмарних послуг або власної компанії, які мають доступ до хмарних систем. Шифрування допомагає запобігти несанкціонованому використанню або витоку даних, навіть якщо ці особи мають доступ до фізичних або логічних ресурсів.

Загалом, методи шифрування поділяються на дві основні категорії: симетричні та асиметричні.

Симетричне шифрування. Це метод, де для шифрування та розшифрування використовується один і той самий секретний ключ. Симетричне шифрування є швидким і ефективним для великих об'ємів даних. Основні алгоритми симетричного шифрування включають:

- AES (Advanced Encryption Standard): Надійний і широко використовуваний алгоритм [14].
- DES (Data Encryption Standard): Старіший стандарт, який вже вважається ненадійним через короткі ключі.
- Triple DES: Покращена версія DES з більшою довжиною ключа.
- Blowfish і Twofish: Інші варіанти симетричних алгоритмів, які забезпечують надійне шифрування.

Асиметричне шифрування. Також відоме як шифрування з відкритим ключем, використовує пару ключів — публічний та приватний. Публічний ключ можна вільно розповсюджувати для шифрування даних, тоді як приватний ключ, який зберігається в таємниці, використовується для розшифрування. Приклади включають:

- RSA: Широко використовуваний для цифрових підписів та шифрування даних [15].

- ECC (Elliptic Curve Cryptography): Забезпечує ту ж безпеку, що й RSA, але з меншими ключами.
- ElGamal: Ще одна система шифрування з відкритим ключем, базована на складних математичних принципах.

AES (Advanced Encryption Standard) є одним з найважливіших сучасних стандартів для симетричного шифрування і має декілька ключових переваг, які забезпечують його актуальність:

- Надійність. AES був прийнятий урядом США та широко використовується у всьому світі для захисту чутливої інформації. Його структура та алгоритми були ретельно перевірені та визнані надійними для забезпечення безпеки даних.
- Стандарт безпеки. AES підтримує кілька довжин ключів (128, 192 та 256 біт), що дозволяє організаціям вибирати рівень безпеки відповідно до їх потреб. Вища довжина ключа забезпечує кращий захист.
- Швидкість та ефективність. AES ефективний як у програмній, так і в апаратній реалізації, забезпечуючи високу швидкість шифрування без значних витрат на продуктивність системи.
- AES може бути використаний у багатьох різних режимах роботи (наприклад, CBC, CTR), що робить його гнучким рішенням для різноманітних застосувань шифрування.

У підсумку, AES залишається золотим стандартом у сфері симетричного шифрування завдяки своїй безпеці, швидкості та гнучкості. Ці якості роблять його ідеальним вибором для захисту даних в корпоративних, урядових та приватних застосуваннях.

Цифровий підпис — це криптографічний механізм, який використовується для перевірки автентичності та забезпечення цілісності даних. Він дозволяє отримувачу переконатися, що документ або повідомлення не було змінено після того, як воно було підписано відправником, і що підпис дійсно належить особі, яка стверджує, що їй належить.

Розглянемо навіщо взагалі потрібний цифровий підпис. Цифровий підпис забезпечує отримувачу доказ, що повідомлення або документ дійсно було створено та відправлено зазначеним відправником. Це важливо в юридичних, фінансових та інших критичних застосуваннях.

Також цифровий підпис гарантує, що зміст повідомлення не був змінений під час передачі. Будь-які зміни у даних, навіть мінімальні, призведуть до помилки під час перевірки підпису.

Крім того, автор підпису не може заперечувати своє авторство повідомлення або документа, тобто підпис забезпечує юридичне підтвердження дій особи.

Розглянемо, як саме відбувається процедура цифрового підписання.

Спочатку генерується пара ключів, що складається з приватного ключа (тримається в таємниці відправником) і публічного ключа (розповсюджується серед потенційних отримувачів). Далі, коли відправник бажає підписати документ або повідомлення, він генерує хеш вмісту, а потім зашифровує цей хеш своїм приватним ключем. Результат цього шифрування стає цифровим підписом, який додається до повідомлення або документа.

Перевірка підпису: Отримувач, використовуючи публічний ключ відправника, розшифровує цифровий підпис для отримання хешу. Потім він самостійно генерує хеш з отриманого повідомлення або документа і порівнює його з розшифрованим хешем. Якщо два хеші збігаються, це підтверджує, що повідомлення не було змінено і що воно дійсно від відправника.

Цифровий підпис є невід'ємною частиною сучасних цифрових комунікацій, юридичних транзакцій та систем електронної комерції, де важливі автентичність, цілісність і невідмовність. Він використовується для забезпечення безпеки в електронних угодах, фінансових транзакціях, управлінні цифровими правами і багатьох інших сферах, де важлива довіра між сторонами.

Звичайно, для доказу авторства необхідно забезпечити індивідуальні облікові записи для кожного користувача, що гарантує відповідальність та

можливість відслідковування дій всередині системи. Кожен користувач, який має доступ до виконання підписаних операцій, повинен мати унікальні реєстраційні дані та особистий ключ, що забезпечують додатковий рівень безпеки та індивідуалізацію доступу до функціоналу цифрового підпису.

Крім основних методів шифрування, важливо впровадити різні рівні безпеки, які зможуть взаємодіяти між собою для забезпечення найвищого рівня захисту:

- Шифрування на рівні додатків: Забезпечує, що всі дані, що обробляються або створюються додатками, автоматично шифруються перед зберіганням або передачею.
- Шифрування на рівні баз даних: Захищає дані, збережені на серверах, незалежно від стану хмарної інфраструктури, з якою вони взаємодіють.

За допомогою сучасних методів шифрування, таких як AES або RSA, та їх інтеграції з цифровими підписами, організації можуть значно покращити захист своїх хмарних ресурсів. Це дозволяє не тільки виконувати нормативні вимоги, але й забезпечує довіру та впевненість клієнтів у тому, що їхні дані обробляються конфіденційно і з належною увагою до безпеки.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Процедура ідентифікації.

Розроблений додаток, що має на меті закрити прогалини, які були розглянуті, написаний на мові програмування Java. Він зустрічає користувача формою, представленою на рисунку 3.1. Для успішного користування додатком, необхідно увійти у систему – пройти процедуру ідентифікації та авторизації, або, якщо Ви новий користувач – виконати реєстрацію натиснувши на відповідну кнопку на панелі. Форма реєстрації представлена на рисунку 3.2.

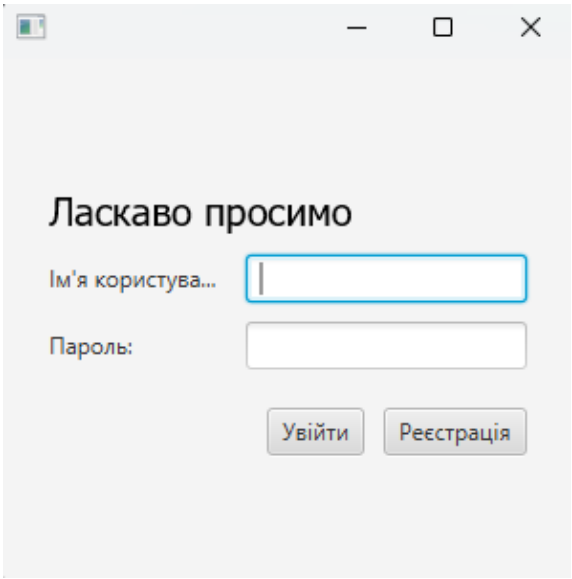


Рисунок 3.1 – Форма для входу

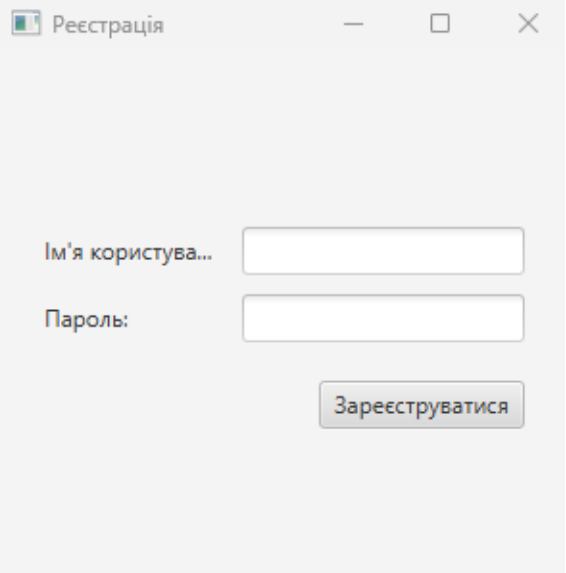


Рисунок 3.2 – Форма реєстрації користувача

Розглянемо детальніше процедуру входу. Дані про користувачів або так званих «юзерів» зберігаються у захищеній базі даних (далі БД) на комп’ютері того, хто користується програмою.

У моєму рішенні було використано реляційну базу даних MySQL. MySQL — це популярна система управління реляційними базами даних, що використовує мову структурованих запитів SQL для управління даними [16]. Основні характеристики:

- Відкритий код: MySQL має відкритий код, і більшість його версій доступні під GNU General Public License (GPL).
- Кросплатформенність: Працює на багатьох операційних системах, включаючи Windows, Linux, macOS та інші Unix-подібні системи.
- Підтримка різних типів даних: Оптимізована для роботи з різними типами даних, такими як числові, дати/часу, рядки, бінарні та інші.
- Масштабованість та гнучкість: Ефективна обробка великих обсягів даних та висока пропускна здатність, з можливістю розширення та кластеризації.
- Безпека: Містить розширені функції безпеки, включаючи шифрування, автентифікацію на основі ролей та інші.

Шлях до цієї бази даних можна визначити вручну. При першому запуску програми та реєстрації у визначеній БД створюється таблиця «users» (якщо вона не була попередньо визначена) і має вигляд, як на рисунку 3.3. Маємо колонки «id» - для пришвидшення виконання запитів, «username», «password», «salt».

	username	password	id	salt
1	Andrii	Z11FQbDBGRH8lXniiTFICrbRhwizWFMGel7aw1GO45Y=	10	JMHQyOGd1NimPovk0sBGjQ==
2	Alex	BSCLw6JLGLoo2mPhV2qEEeosLacLQBZQamkvZEeNeBI=	11	Z1UVyT6Mr82v7laoITTpxQ==
3	Max	eudvQfuWnnV/CDM0fyn4to14YrNS2P+kOmir7Cr/yi4=	12	IqS1L2oNQFBBoNHUGBBu2g==
4	Pete	K4QMBY6+bDLYFzi5kAllZgnlFHmxqbTedk8F3lwbbww=	13	Cr0uiT1o+8YEBpdT6M3T7A==

Рисунок 3.3 – Таблиця користувачів

Звичайно, для надійності ми не можемо зберігати паролі у звичайному вигляді (як його ввів користувач). Тому вони зберігаються у вигляді хешу з використанням так званої «солі», ускладнюючи атаки грубою силою або використання готових таблиць хешів. Сіль (salt) — це випадково згенерована послідовність, яка додається до пароля перед хешуванням. Кожен користувач має унікальну сіль, що зберігається разом із хешем пароля в базі даних і генерується при реєстрації. На лістингу 3.1 бачимо, яким чином генерується хешування паролів.

Лістинг 3.1.1. Метод хешування паролю

```
public static String hashPassword(String password, String salt) throws
NoSuchAlgorithmException {
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    // Додаємо сіль до пароля перед хешуванням
    String saltedPassword = salt + password;
    byte[] hashedBytes = md.digest(saltedPassword.getBytes());
    return Base64.getEncoder().encodeToString(hashedBytes);
}
```

Сам процес входу відбувається за допомогою функції authenticateUser() (Лістинг 3.2), суть якої перевірити хеш пароля, якій був введений з реальним, та взагалі дізнатися чи існує юзер із заданим ім’ям. У разі невдалого входу користувач побачить відповідне повідомлення (рисунок 3.4).

Лістинг 3.1.2. Метод аутентифікації користувача

```
public boolean authenticateUser(String username, String password) {
    String sql = "SELECT password, salt FROM Users WHERE username = ?";
    try (Connection conn = this.connect();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, username);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            String storedHash = rs.getString("password");
            String salt = rs.getString("salt");
            String hashedPassword = CryptoUtils.hashPassword(password, salt); // Хешування введеного пароля з використанням збереженої солі
            if (hashedPassword.equals(storedHash)) {
                // Користувач аутентифікований
                return true;
            }
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    }
    return false;
}
```

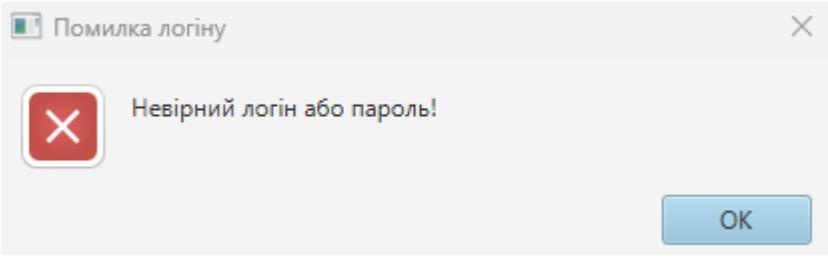


Рисунок 3.4 – Повідомлення про помилку входу

Під час реєстрації користувач не може використати існуюче ім’я – воно є унікальним. У разі дублювання логіну з’явиться повідомлення, інформуючи про це (рисунок 3.5).

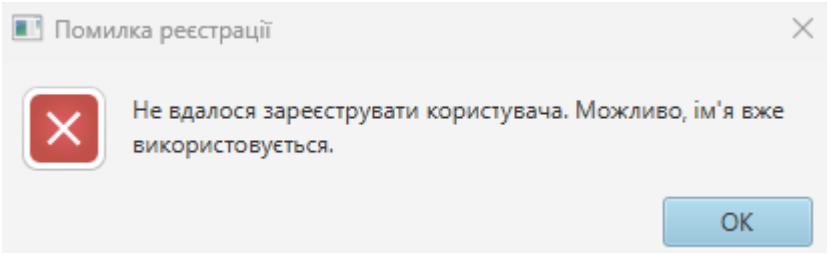


Рисунок 3.5 – Повідомлення про помилку реєстрації

Якщо процедура авторизації пройшла успішно, користувач побачить панель, представлену на рисунку 3.6.

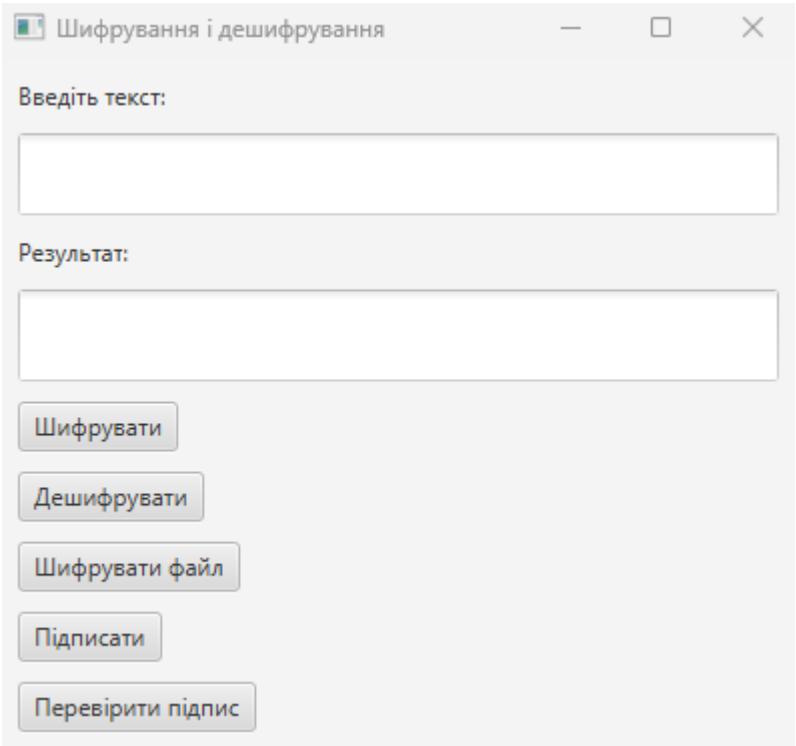


Рисунок 3.6 – Основна панель програми

З даного вікна можна побачити наступні кнопки:

- «Шифрувати» - процедура шифрування повідомлення;
- «Дешифрувати» - обернена процедура шифрування;
- «Шифрувати файл» - перехід до іншого вікна для роботи з файлами;

- «Підписати» - використати цифровий підпис;
- «Перевірити підпис» - перевірити цифровий підпис.

При натисканні на кнопку «Шифрувати файл» буде відкрито нове вікно, представлене на рисунку 3.7. Воно є необхідним для виконання процедур із файлами. Кнопки представлені у даному вікні:

- «Оберіть файл» - вибір Вашого файлу, який необхідно захистити;
- «Зашифрувати» - шифрування файлу;
- «Розшифрувати» - обернена операція шифрування файлу;
- «Підписати» - застосувати цифровий підпис на документ;
- «Перевірити підпис» - перевірка дійсності підпису.

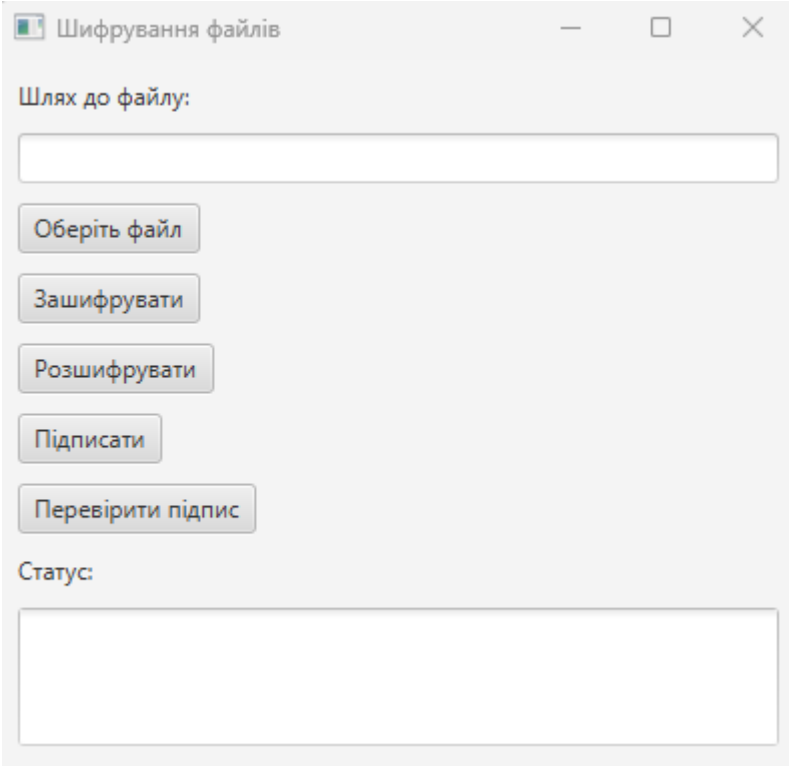


Рисунок 3.7 – Панель роботи з файлами

3.2 Шифрування.

У разі несанкціонованого доступу зловмисників до даних у хмарі, або у разі несподіваного витоку інформації, шифрування може стати гарною завадою та не дасть миттєво розповсюдити файли у загальний доступ. Тому непоганим рішенням може бути шифрування даних перед зберіганням у хмарі.

Розглянемо реалізацію даного функціоналу у додатку. Функції шифрування файлу та тексту є різними і представлені на лістингах 3.2.1 та 3.2.2 відповідно.

Лістинг 3.2.1. Метод шифрування файлу

```
private void encryptFile(File inputFile, File outputFile) throws IOException,
NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException,
BadPaddingException, IllegalBlockSizeException {
    Cipher cipher = Cipher.getInstance("AES");
    SecretKey secretKey = chooseKeyDialog();
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    try (FileInputStream inputStream = new FileInputStream(inputFile);
         FileOutputStream outputStream = new FileOutputStream(outputFile)) {
        byte[] inputBytes = new byte[(int) inputFile.length()];
        inputStream.read(inputBytes);
        byte[] encryptedBytes = cipher.doFinal(inputBytes);
        outputStream.write(encryptedBytes);
    }
  }
```

Лістинг 3.2.2 Метод шифрування тексту

```
public static String encryptText(String data, SecretKey secretKey) throws
NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException,
IllegalBlockSizeException, BadPaddingException {
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    byte[] encryptedData = cipher.doFinal(data.getBytes());
    return Base64.getEncoder().encodeToString(encryptedData);
}
```

У даному коді:

- File inputFile - Це об'єкт файлу, який містить дані, які потрібно зашифрувати.

- File outputFile - Це об'єкт файлу, у який будуть записані зашифровані дані.

Функція оголошує, що може виникнути кілька типів виключень, які можуть бути викликані різними частинами процесу шифрування:

- IOException: Може виникнути під час читання вхідного файлу або запису вихідного файлу.
- NoSuchPaddingException, NoSuchAlgorithmException: Виникають, якщо вказане шифрування або його параметри не підтримуються.
- InvalidKeyException, BadPaddingException, IllegalBlockSizeException: Можуть виникнути в процесі ініціалізації шифру або під час виконання фіналізації шифрування.

a) Cipher cipher = Cipher.getInstance("AES") - Ініціалізує об'єкт Cipher для шифрування, вказуючи на використання алгоритму AES. AES (Advanced Encryption Standard) — це широко використовуваний алгоритм симетричного шифрування.

b) Для використання даного алгоритму необхідно використовувати ключ, який генеруємо, або дістаємо інший.

SecretKey secretKey = chooseKeyDialog() - Ця функція відображає діалогове вікно для вибору або генерації секретного ключа, потрібного для шифрування. Ключ повинен бути сумісний з вибраним алгоритмом шифрування (AES).

c) Ініціалізація шифрування:

cipher.init(Cipher.ENCRYPT_MODE, secretKey): Ініціалізує шифр у режимі шифрування, використовуючи отриманий секретний ключ.

d) Читання та шифрування даних:

Відкриття потоків FileInputStream для читання даних з вхідного файлу та FileOutputStream для запису зашифрованих даних у вихідний файл.

e) Читання даних з вхідного файлу в масив байтів inputBytes.

f) Виконання шифрування даних методом cipher.doFinal(inputBytes), що повертає зашифровані дані в масиві encryptedBytes.

g) Запис зашифрованих байтів в вихідний файл.

h) Закриття ресурсів.

Використання конструкції try-with-resources (try (...) {}) гарантує, що всі використані ресурси (потоки вводу/виводу) будуть належним чином закриті після завершення роботи блоку коду, незалежно від того, чи було виконання успішним, чи сталася помилка.

Як вже зазначалося, метод chooseKeyDialog() (код якого можна знайти у додатку) використовується для визначення секретного ключа. Розглянемо детальніше цю функцію.

Під час виконання відповідних операцій з даними користувачу буде запропоновано наступні опції (представлено на рисунку 3.8).

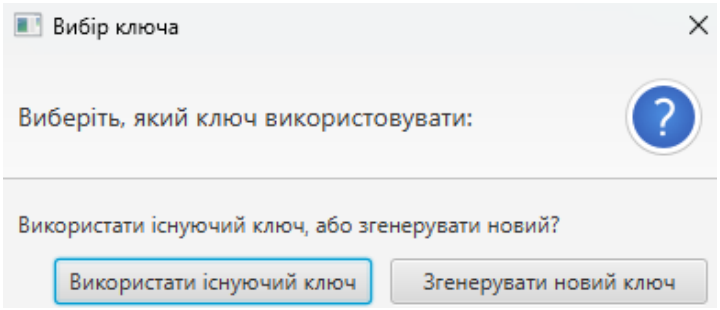


Рисунок 3.8 – Вибір ключа для шифрування

Для генерації ключа необхідно ввести кодове слово, за яким потім цей ключ можна використовувати (рисунок 3.9). Створені дані зберігаються у тій самій БД у таблиці «keys» і мають вигляд як на рисунку 3.10.

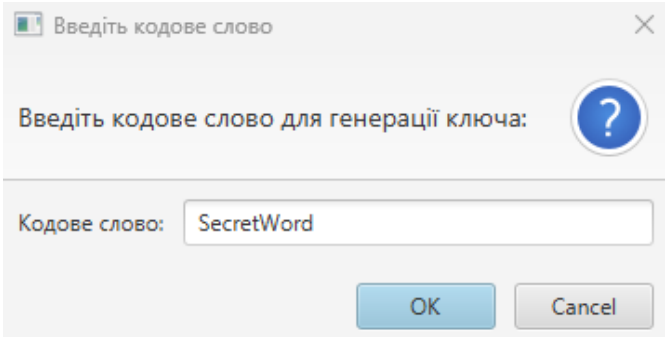


Рисунок 3.9 – Генерація ключа

id	key_data	secret_word	user_id
10	0xF869AC8570A1F73B9D9FECC3EA41FA5A	6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b	7
11	0xD2C28D83E6B5E4C6D27790C61811454C	gB6Nuu1v23EuJDdtQNKMzOU9OJumm5A8UX85k9wsV8A=	7

Рисунок 3.10 – Таблиця з ключами

Якщо користувач хоче використовувати вже існуючий ключ, який був створений під час попередніх сесій, йому буде запропоновано використати кодове слово. У разі невдалої спроби знайти відповідні дані у БД, система виведе відповідне повідомлення (рисунок 3.11) і запропонує спробувати ще раз.

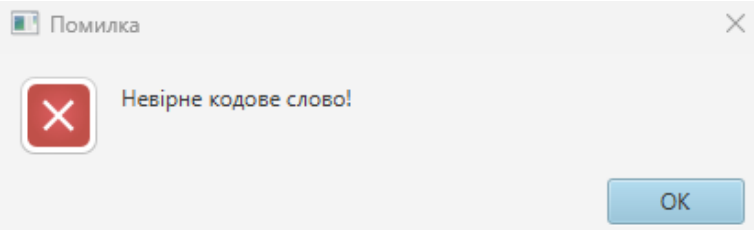


Рисунок 3.11 – Помилка при невірному кодовому слові

Розглянемо приклад шифрування даних. Для початку зашифруємо просте повідомлення з головного вікна. Результат виконання можна побачити на рисунку 3.12.

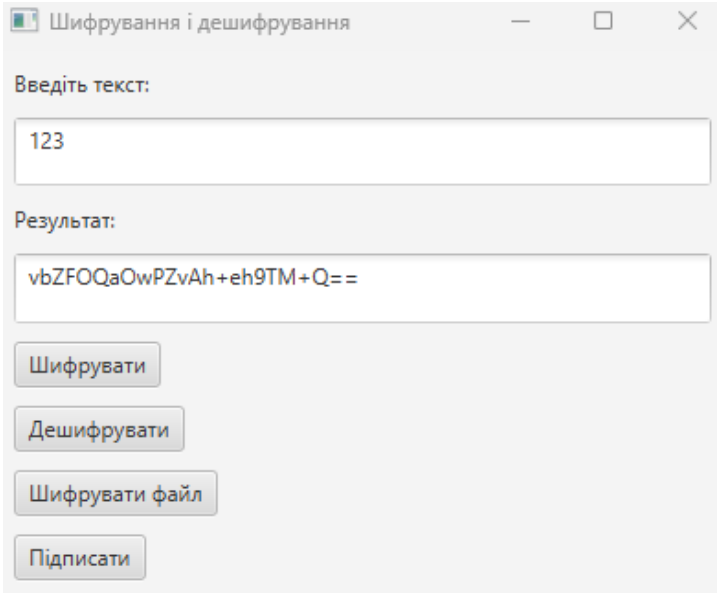


Рисунок 3.12 – Результат шифрування повідомлення

Для шифрування файлу треба перейти до відповідного вікна та вибрати потрібний файл. Зашифрований варіант буде створено у тому самому місці з

ім’ям оригінала з префіксом encrypted_ (encrypted_оригінальна_назва). Результат виконання представлено на рисунку 3.13.

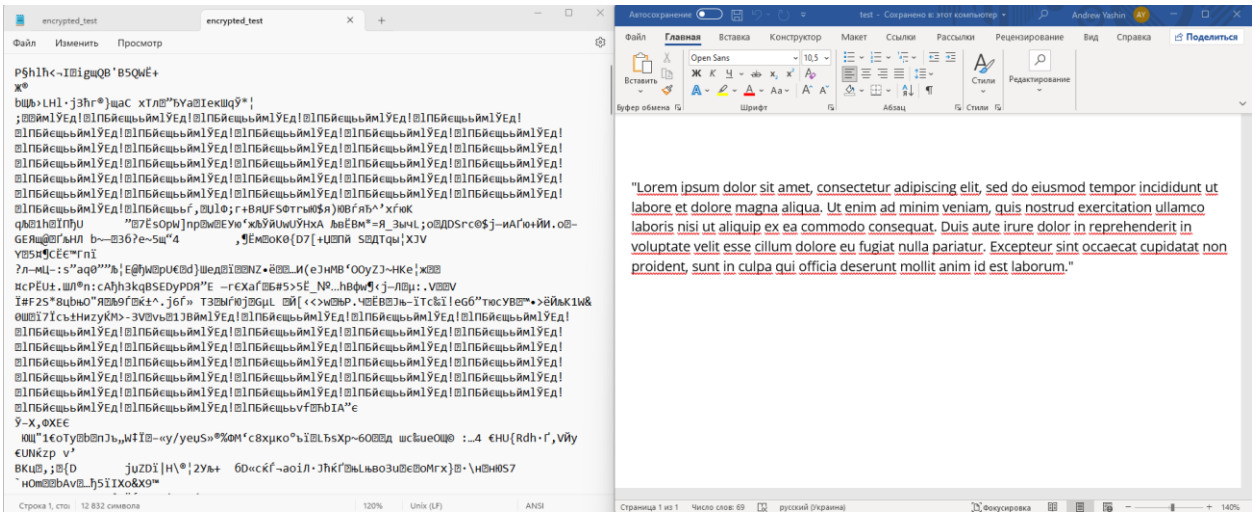


Рисунок 3.13 – Результат шифрування файлу

Зашифровані дані це звичайно круто та корисно у разі втрати, однак обернена процедура є очевидно необхідною. Додаток пропонує наступну реалізацію дешифрування, представлену на лістингах 3.2.3 та 3.2.4.

Лістинг 3.2.3. Метод дешифрування тексту

```
public static String decryptText(String encryptedData, SecretKey secretKey) throws BadPaddingException {
    try {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedData = cipher.doFinal(Base64.getDecoder().decode(encryptedData));
        return new String(decryptedData);
    } catch (BadPaddingException e) {
        // Обробка помилки при неправильному дешифруванні
        System.err.println("Помилка дешифрування: неправильні дані або ключ.");
        e.printStackTrace();
        Dialog.showAlert("Помилка", "Дешифрування неможливе!", Alert.AlertType.ERROR);
        return null;
```

```
    } catch (Exception e) {
        // Обробка інших винятків, які можуть виникнути під час дешифрування
        System.err.println("Помилка дешифрування: " + e.getMessage());
        e.printStackTrace();
        return null;
    }
}
```

Лістинг 3.2.4 Метод дешифрування файлу

```
private void decryptFile(File inputFile, File outputFile) throws IOException,
NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException,
BadPaddingException, IllegalBlockSizeException {
    Cipher cipher = Cipher.getInstance("AES");
    TextInputDialog dialog = new TextInputDialog();
    dialog.setTitle("Введіть кодове слово");
    dialog.setHeaderText("Введіть кодове слово для використання існуючого
ключа:");
    dialog.setContentText("Кодове слово:");
    Optional<String> inputSecretWord = dialog.showAndWait();
    if (inputSecretWord.isPresent()) {
        // Перевірка, чи введене кодове слово вірне
        String word = inputSecretWord.get();
        if (keyStorage.getSecretKeyByWord(word) != null) {
            // Повертаємо існуючий ключ
            cipher.init(Cipher.DECRYPT_MODE,
keyStorage.getSecretKeyByWord(word));
        } else {
            Dialog.showAlert("Помилка", "Невірне кодове слово!",
Alert.AlertType.ERROR);
        }
    }
    try (FileInputStream inputStream = new FileInputStream(inputFile);
```

```
         FileOutputStream outputStream = new FileOutputStream(outputFile)) {
        byte[] inputBytes = new byte[(int) inputFile.length()];
        inputStream.read(inputBytes);
        byte[] decryptedBytes = cipher.doFinal(inputBytes);
        outputStream.write(decryptedBytes);
    }
}
```

Метод decryptFile призначений для дешифрування файлів, використовуючи симетричний алгоритм шифрування AES, а саме для відновлення оригінальних даних з файлів, які були зашифровані. Ось детальний опис процесу, який відбувається в цьому методі:

Параметри:

- File inputFile: Файл, що містить зашифровані дані.
- File outputFile: Файл, у який будуть записані розшифровані дані.

Метод декларує кілька типів виключень, які можуть бути викликані під час дешифрування.

a) Cipher cipher = Cipher.getInstance("AES"): Цей рядок ініціалізує об'єкт Cipher, який використовується для дешифрування, вказуючи використання алгоритму AES.

b) Отримання ключа через діалогове вікно:

TextInputDialog dialog = new TextInputDialog(): Створюється діалогове вікно для введення секретного слова, яке потрібне для отримання ключа дешифрування.

c) Якщо користувач вводить слово і воно вірне, використовується метод keyStorage.getSecretKeyByWord(word) для отримання відповідного секретного ключа.

d) Ініціалізація дешифрування:

cipher.init(Cipher.DECRYPT_MODE, keyStorage.getSecretKeyByWord(word)): Шифр ініціалізується в режимі дешифрування з отриманим ключем.

e) Читання та дешифрування даних:

Дані читаються з файлу за допомогою FileInputStream.

Дешифрування виконується методом cipher.doFinal(inputBytes), який перетворює зашифровані байти назад у їх оригінальний вигляд.

f) Результати записуються у вихідний файл через FileOutputStream.

g) Закриття ресурсів:

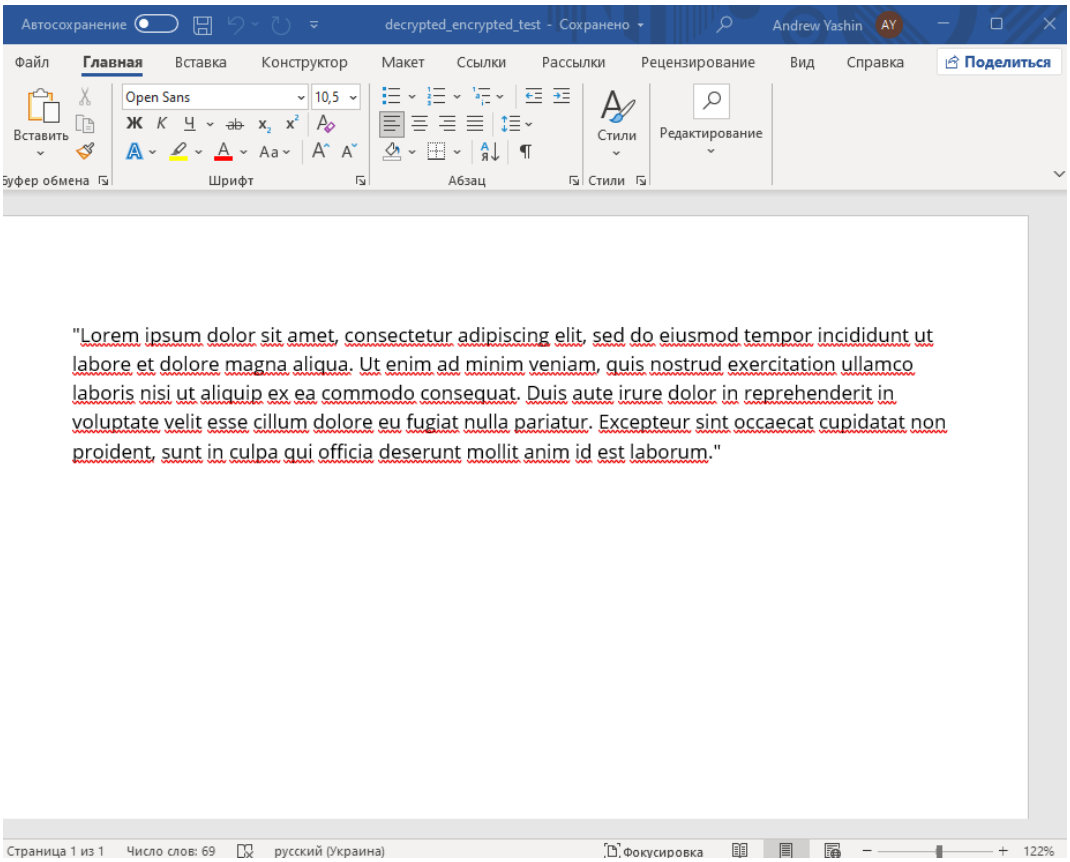
Конструкція try-with-resources гарантує, що всі потоки (FileInputStream, FileOutputStream) будуть належно закриті після завершення операцій читання та запису, незалежно від того, чи виконання коду було успішним чи ні.

h) Обробка помилок:

Якщо користувач вводить невірне кодове слово, відображається діалогове вікно з помилкою, що інформує про помилкове введення. Це забезпечує захист від невірного доступу до зашифрованих даних.

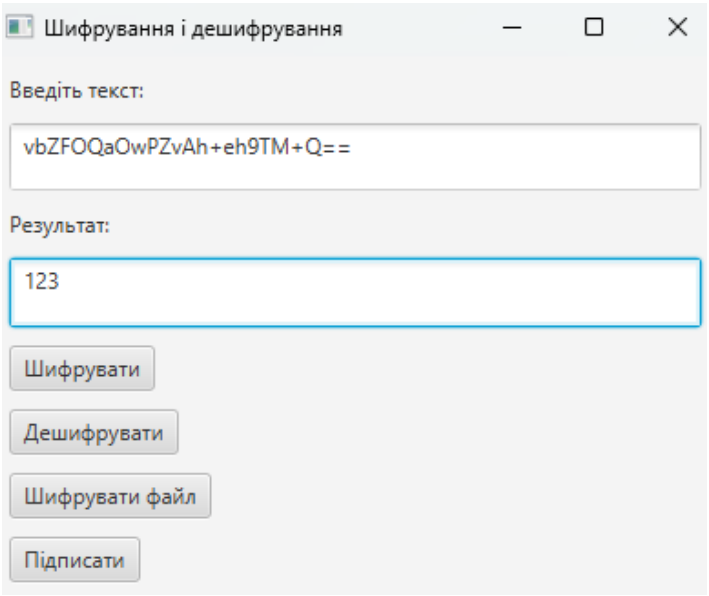
Цей метод ефективно забезпечує безпеку конфіденційних даних, вимагаючи від користувача верифікації для доступу, і забезпечує зворотнє відновлення оригінальних даних з зашифрованої форми.

Для дешифрування користувачу необхідно вибрати файл та натиснути кнопку «Розшифрувати» та ввести кодове слово для використання згенерованого ключа. У разі вірного вводу кодового слова, результатом дешифрування згенерованого файлу encrypted_test буде новий файл decrypted_ encrypted_test з оригінальним вмістом (рисунок 3.14).



3.14 – Результат дешифрування файлу

На рисунку 3.15 результат виконання дешифрування звичайного тексту.



3.15 – Результат дешифрування повідомлення

Клас Cipher у Java є основним компонентом Java Cryptography Architecture (JCA) і використовується для забезпечення криптографічних

функцій шифрування та дешифрування [18]. Існує декілька причин, чому саме Cipher використовується в методі encryptText для шифрування даних.

Він надає універсальний інтерфейс для реалізації різноманітних алгоритмів шифрування. Це означає, що ви можете використовувати той самий API для шифрування даних з використанням різних криптографічних алгоритмів, таких як AES, DES, RSA та інші, просто змінюючи рядок конфігурації.

Також Cipher дозволяє вибирати не тільки алгоритм шифрування, але й режими роботи (наприклад, ECB, CBC, CFB) та схеми падінгу (наприклад, PKCS5Padding, NoPadding). Це робить Cipher надзвичайно гнучким інструментом для криптографічних операцій, дозволяючи точно налаштувати параметри шифрування відповідно до конкретних вимог безпеки.

Крім того використання класу Cipher допомагає забезпечити високий рівень безпеки, оскільки він входить до стандартного Java API і підтримується і перевіряється численними розробниками та аудиторами безпеки. Це зменшує ризики, пов'язані з реалізацією власних криптографічних алгоритмів.

Також Cipher дозволяє виконувати шифрування та дешифрування в кілька простих кроків, зменшуючи складність коду і забезпечуючи чистий, зрозумілий інтерфейс для роботи з криптографією.

Клас Cipher інтегрований з іншими частинами Java Security, такими як SecretKey, KeyGenerator, KeyPairGenerator, KeyStore і так далі, забезпечуючи плавне взаємодію між різними компонентами криптографічної системи.

Завдяки своїй універсальності, підтримці стандартів, безпеці та легкості використання, клас Cipher є відмінним вибором для реалізації шифрування в Java-додатках. Він надає розробникам потужний інструмент для впровадження надійних криптографічних функцій без потреби у глибокому розумінні нюансів кожного криптографічного алгоритму.

3.3 Цифровий підпис.

Цифровий підпис є важливим інструментом у сучасному цифровому світі, зокрема для забезпечення безпеки та довіри до електронних документів, програмного забезпечення та інших даних.

В представленому додатку існує можливість скористатися цифровим підписом як для простого тексту, що не є вкрай популярним, так і для файлів – популярна та корисна практика. За виконання цієї операції відповідає функція signFile(), яка представлена на лістингу 3.2.5.

Лістинг 3.3.1. Метод підпису файлу

```
public void signFile() {
    try {
        String filePath = filePathTextField.getText();
        File inputFile = new File(filePath);
        if (!inputFile.exists()) {
            outputTextArea.setText("Файл не знайдено.");
            return;
        }
        byte[] fileData = Files.readAllBytes(inputFile.toPath());
        KeyPair keyPair = CryptoUtils.generateKeyPair();
        byte[] signature = CryptoUtils.signData(fileData, keyPair.getPrivate());
        String signatureBase64 = Base64.getEncoder().encodeToString(signature);
        outputTextArea.setText("Підписана інформація: " + inputFile.getName() +
"\nПідпис: " + signatureBase64);
        keyStorage.saveFileSignature(inputFile.getName(), fileData, signatureBase64,
Base64.getEncoder().encodeToString(keyPair.getPublic().getEncoded()));
    } catch (Exception ex) {
        Dialog.showAlert("Помилка", "Помилка при створенні підпису: " +
ex.getMessage(), Alert.AlertType.ERROR);
        ex.printStackTrace();
    }
}
```

Детально розглянемо кожен крок, що виконується в цій функції:

a) Читання вмісту файлу

- Отримання шляху до файлу: Шлях до файлу зчитується з текстового поля filePathTextField, яке є частиною графічного інтерфейсу користувача.
- Перевірка наявності файлу: Створюється об'єкт File, і перевіряється його існування. Якщо файл не знайдено, виводиться повідомлення у текстове поле outputTextArea і вихід з функції.
- Читання даних файлу: Дані файлу читаються у вигляді масиву байтів за допомогою методу Files.readAllBytes().

b) Генерація та застосування цифрового підпису

- Генерація ключової пари: Викликається метод CryptoUtils.generateKeyPair(), який генерує пару криптографічних ключів (приватний та публічний).
- Створення цифрового підпису: Дані файлу підписуються використанням приватного ключа через метод CryptoUtils.signData(). Цей метод використовує алгоритм підпису, такий як SHA з RSA, для створення цифрового підпису, який буде розглянутий далі.
- Кодування підпису в Base64: Цифровий підпис перетворюється в строку формату Base64, що зручно для зберігання та відображення. Base64 — це широко використовувана схема кодування, що дозволяє перетворювати бінарні дані (наприклад, зображення або файлі) у ASCII-рядки. Це корисно для передачі бінарних даних через медіа, які призначені для роботи тільки з текстом, такі як електронна пошта чи веб-сторінки.

c) Зберігання підпису та інформації про файл

- Виведення інформації про підпис: У текстове поле outputTextArea виводиться інформація про назву файлу та його підпис у форматі Base64.

- Зберігання підпису та даних: Метод keyStorage.saveFileSignature() використовується для зберігання інформації про файл, його дані, підпис та публічний ключ у кодуванні Base64. Цей крок може включати запис у базу даних або іншу форму зберігання.

d) Обробка виключень

- Якщо в процесі виконання функції виникають помилки, вони перехоплюються блоком catch, виводиться повідомлення про помилку, а також виконується ex.printStackTrace() для детального логування помилки.

Сам процес підпису відбувається у функції signData(), тож розглянемо її детальніше.

Лістинг 3.3.2. Метод підпису даних

```
public static byte[] signData(String data, PrivateKey privateKey) throws Exception {
    Signature signature = Signature.getInstance("SHA256withRSA");
    signature.initSign(privateKey);
    signature.update(data.getBytes());
    return signature.sign();
}
```

a) Параметри

String data: Вхідні дані, які потрібно підписати. Це може бути будь-який текст, наприклад, повідомлення, документ або інша інформація.

PrivateKey privateKey: Приватний ключ, що використовується для генерації підпису. Ключ повинен бути сгенерований та збережений безпечним чином.

b) Процес

Ініціалізація об'єкта підпису:

Signature signature = Signature.getInstance("SHA256withRSA"): Ця команда створює екземпляр об'єкта Signature, який використовує алгоритм SHA-256 з RSA для генерації підпису. SHA-256 є функцією хешування, яка забезпечує високий рівень безпеки, а RSA — це алгоритм асиметричного шифрування.

Ініціалізація підпису приватним ключем:

signature.initSign(privateKey): Цей метод ініціалізує підпис за допомогою приватного ключа. Це необхідно для того, щоб можна було використовувати приватний ключ для генерації цифрового підпису.

Оновлення підпису з даними:

signature.update(data.getBytes()): Метод update використовується для додавання даних, які потрібно підписати, до об'єкта Signature. data.getBytes() перетворює строку data в масив байтів, що є вхідними даними для процесу підпису. Важливо, що використання однакової кодувальної схеми є критичним при перетворенні строки у байти для забезпечення консистенції між різними системами.

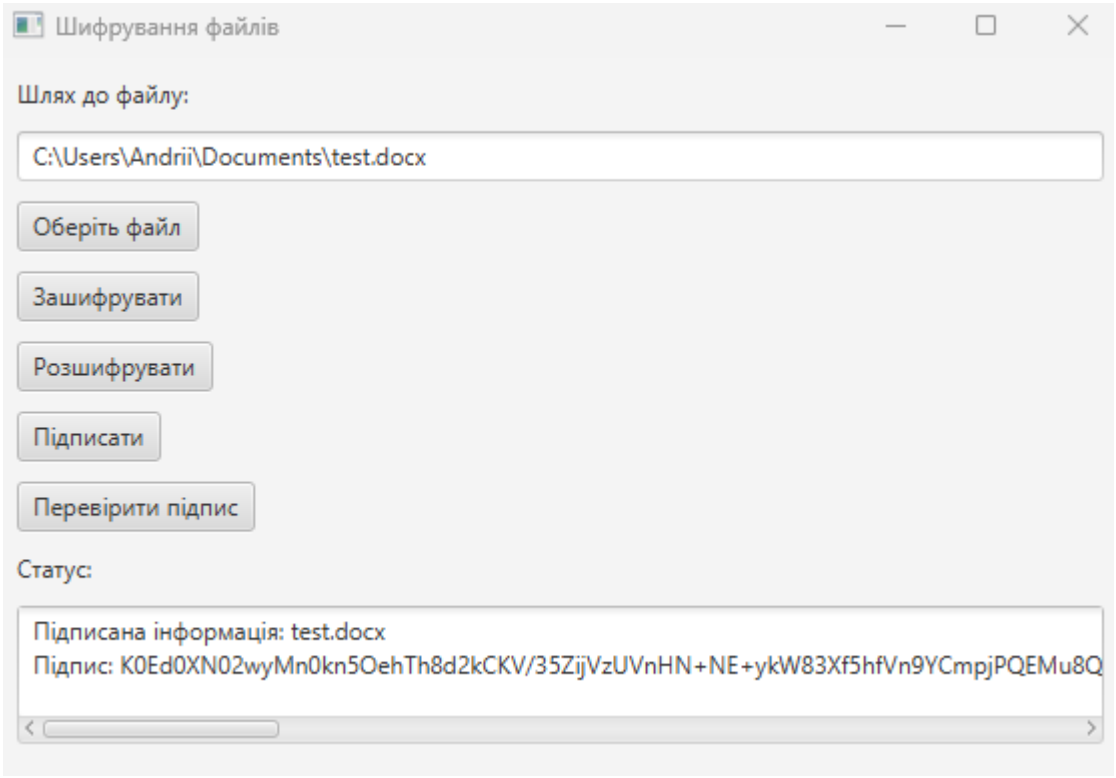
Генерація підпису:

return signature.sign(): Після того як всі дані були додані в об'єкт Signature, викликається метод sign(), який генерує фінальний цифровий підпис. Цей підпис повертається у вигляді масиву байтів. Цей підпис може бути перевірений будь-ким, хто має доступ до відповідного публічного ключа, щоб підтвердити, що дані не були змінені після підпису та що вони були підписані власником приватного ключа.

Виключення:

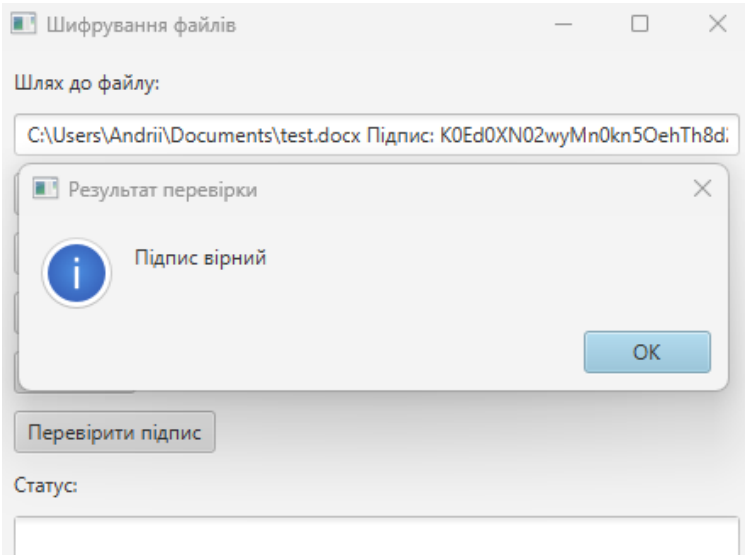
Функція декларує, що може викидати Exception, що означає, що будь-які винятки, пов'язані з процесом підпису, мають бути оброблені вищими рівнями виклику функції.

Для користувача процес підпису дуже схожий на процес шифрування. Необхідно також вибрати файл (або просто ввести дані) та натиснути кнопку «Підписати». Після цього користувач побачить відповідне повідомлення. Приклад представлено на рисунку 3.16.



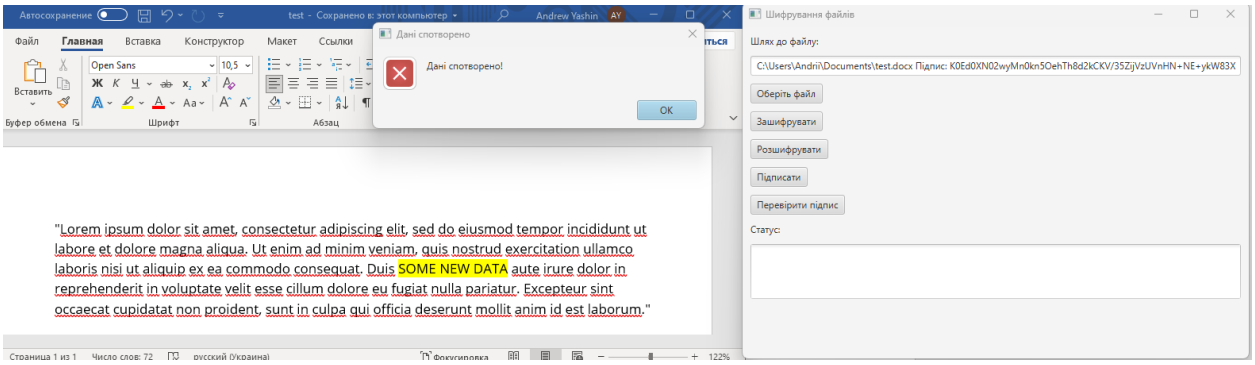
3.16 – Результат підпису файлу

Для перевірки самого підпису необхідно у полі для вводу ввести шлях до потрібного файлу, сам підпис та натиснути «Перевірити підпис». У разі позитивного результату (якщо підпис вірний та файл не було змінено) користувач побачить відповідне повідомлення (рисунок 3.17).

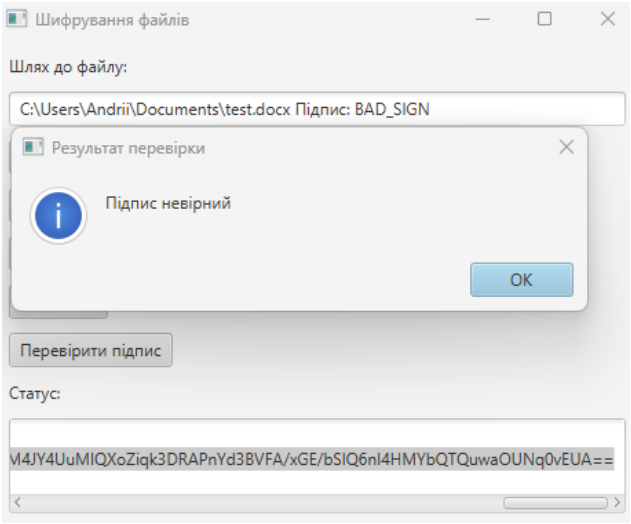


3.17 – Результат вдалої перевірки підпису файлу

Однак у разі зміни файлу (рисунок 3.18) або неправильного підпису (3.19) операція буде неуспішною та видасть відповідне повідомлення. Змінені дані на рисунках виділені жовтим кольором.



3.18 – Повідомлення при зміні файлу



3.19 – Результат невдалої перевірки підпису файлу

Розглянемо яким чином виконується перевірка підпису.

Лістинг 3.3.3. Метод отримання даних для перевірки підпису

```
public void verifyFileSignature() {
    String[] parts = filePathTextField.getText().split(" Підпис: ");
    System.out.println(Arrays.toString(parts));
    if (parts.length < 2) {
        Dialog.showAlert("Помилка", "Недостатньо даних для перевірки підпису", Alert.AlertType.ERROR);
        return;
    }
    String signature = parts[1];
```

```
String filePath = parts[0].replace("Підписана інформація: ", "");
File inputFile = new File(filePath);
if (!inputFile.exists()) {
    outputTextArea.setText("Файл не знайдено.");
    return;
}
try {
    byte[] fileData = Files.readAllBytes(inputFile.toPath());
    boolean isValid = keyStorage.verifyFileSignature(fileData,
inputFile.getName(),signature);
    Dialog.showAlert("Результат перевірки", "Підпис " + (isValid ? "вірний" :
"невірний"), Alert.AlertType.INFORMATION);
} catch (IOException e) {
    Dialog.showAlert("Результат перевірки", "Підпис невірний",
Alert.AlertType.INFORMATION);
    throw new RuntimeException(e);
} catch (IllegalArgumentException e) {
    Dialog.showAlert("Результат перевірки", "Підпис невірний",
Alert.AlertType.INFORMATION);
}}
```

Цей метод verifyFileSignature() призначений для перевірки цифрового підпису файлу, використовуючи текстовий інтерфейс та логіку в системі для перевірки цих підписів. Давайте розглянемо детально кожен крок у цій функції:

a) Розділення вхідних даних:

- Строка, отримана з filePathTextField, розділяється за допомогою роздільника " Підпис: ". Це передбачає, що введення містить ім'я файлу, за яким слідує його підпис.

Результат розбиття зберігається у масиві parts.

b) Перевірка на наявність необхідних компонентів:

- Перевіряється, чи масив parts містить принаймні два елементи (ім'я файлу та підпис). Якщо ні, відображається помилка про недостатність даних для перевірки підпису, і метод завершується.

c) Отримання шляху до файлу та підпису:

- Шлях до файлу витягується з першого елементу масиву, попередньо видаляючи текст "Підписана інформація: ", якщо він є.
- Підпис зберігається як другий елемент масиву.

d) Перевірка на існування файлу:

- Створюється об'єкт File зі шляхом до файлу, і перевіряється, чи файл існує. Якщо файл не знайдено, виводиться відповідне повідомлення і метод завершується.

e) Читання даних файлу:

- Зчитування байтів файлу за допомогою Files.readAllBytes(), що перетворює вміст файлу в масив байтів.

f) Перевірка підпису:

- Викликається метод keyStorage.verifyFileSignature() з переданими даними файлу, ім'ям файлу та підписом. Метод повертає boolean, який вказує на те, чи є підпис вірним.
- В залежності від результату перевірки виводиться відповідне повідомлення.

g) Обробка виключень.

Сама процедура перевірки виконується в методі verifyFileSignature, показаний на лістингу 3.3.4.

Лістинг 3.3.4. Метод перевірки підпису

```
public boolean verifyFileSignature(byte[] data, String fileName, String signatureBase64)
{
    String sql = "SELECT data_hash, signature, public_key FROM FileSignatures
WHERE user_id = ? AND id = (SELECT MAX(id) FROM FileSignatures WHERE
user_id = ? AND file_name = ?)";
    try (Connection conn = connect();
```

```
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, EncryptionApp.userId);
            pstmt.setInt(2, EncryptionApp.userId);
            pstmt.setString(3, fileName);
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    String dataHashBase64 = rs.getString("data_hash");
                    //String signatureBase64 = rs.getString("signature");
                    System.out.println(rs.getString("signature").equals(signatureBase64));
                    String publicKeyBase64 = rs.getString("public_key");
                    MessageDigest digest = MessageDigest.getInstance("SHA-256");
                    byte[] hash = digest.digest(data);
                    byte[] dataHash = Base64.getDecoder().decode(dataHashBase64);
                    if(!Arrays.equals(dataHash, hash)) {
                        Dialog.showAlert("Дані спотворено", "Дані спотворено!",
Alert.AlertType.ERROR);
                        return false;
                    }
                    byte[] signature = Base64.getDecoder().decode(signatureBase64);
                    byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyBase64);
                    PublicKey publicKey = getPublicKeyFromBytes(publicKeyBytes);
                    return verifyDigitalSignature(data, signature, publicKey);
                }
            }
        } catch (SQLException | NoSuchAlgorithmException | InvalidKeyException |
                 SignatureException e) {
            e.printStackTrace();
        }
        return false;
    }
```

Відкривається з'єднання з базою даних і готується SQL запит. Запит виконується, і результати зберігаються в об'єкті ResultSet. Якщо в базі знайдено запис, зчитуються хеш даних, підпис і публічний ключ. Хеш даних, що збережений в базі, порівнюється з новоствореним хешем вхідних даних (SHA-256). Якщо вони не збігаються, виводиться повідомлення про спотворення даних і метод повертає false.

Слід зазначити, чому було використано саме такий спосіб. Клас Signature у Java є частиною Java Cryptography Architecture (JCA) і спеціально призначений для створення та верифікації цифрових підписів [17].

Він був розроблений як високорівневий інструмент для роботи з цифровими підписами, забезпечуючи всі необхідні функції для їх створення та перевірки. Це забезпечує, що розробники можуть використовувати готовий, надійно тестований компонент замість самостійного впровадження складних криптографічних алгоритмів.

Використання стандартного класу з Java API гарантує високий рівень безпеки, оскільки ці класи ретельно перевіряються та оновлюються для захисту від відомих уразливостей. Робота з класом Signature зменшує ризик помилок, які можуть призвести до слабких місць у системі безпеки додатків.

Клас Signature підтримує різноманіття криптографічних алгоритмів підпису, таких як SHA1withRSA, SHA256withRSA, і багато інших. Це дає можливість легко змінювати алгоритми підпису без зміни базової логіки програми.

Signature API дозволяє легко інтегрувати процес підпису у програми, забезпечуючи прості методи для ініціалізації, оновлення даних, підписування та верифікації підписів. Розробники можуть зосередитись на бізнес-логіці, не турбуючись про внутрішні деталі криптографічного процесу.

Signature взаємодіє з іншими компонентами JCA, такими як KeyPairGenerator, KeyFactory, і CertificateFactory. Це дозволяє легко використовувати цифрові сертифікати, генерувати ключі та інші пов'язані завдання в єдиному криптографічному контексті.

Підпис декодується з Base64 та перевіряється з використанням публічного ключа. Публічний ключ декодується з Base64 і перетворюється у формат, придатний для перевірки підпису. Використовується метод verifyDigitalSignature, який повертає true, якщо підпис вірний, або false, якщо ні.

Весь код додатку можна побачити у додатку А, або на моєму GitHub [20]. Там також можна побачити інструкцію щодо встановлення даної програми.

ВИСНОВКИ

Ця кваліфікаційна робота мала на меті розробку та аналіз ефективних програмних методів, які забезпечують протидію спотворенню даних у хмарних сервісах. Основним фокусом було дослідження та оптимізація методів шифрування та використання цифрових підписів, що забезпечують збереження конфіденційності, цілісності та доступності даних.

Протягом роботи було реалізовано та досліджено кілька новітніх алгоритмів шифрування, які виявилися ефективними у протистоянні загрозам безпеки даних у хмарних системах. Використання цифрових підписів значно підвищило здатність системи ідентифікувати та запобігати несанкціонованим змінам у даних, а також забезпечило важливий механізм для підтвердження автентичності джерела інформації.

Значний внесок роботи полягає у підвищенні ефективності виявлення та відповіді на інциденти безпеки у хмарних сервісах, що стало можливим завдяки інтеграції передових технологій криптографічного захисту. Ці технології дозволяють користувачам хмарних сервісів мати більшу впевненість у тому, що їхні дані обробляються та зберігаються з належним рівнем безпеки та не покладатися на «чорний ящик».

Викладені у даній роботі методи та техніки можуть бути використані для покращення безпекових практик у широкому спектрі галузей, де використовуються хмарні обчислення, включаючи фінанси, охорону здоров'я та урядові служби. Результати дослідження підкреслюють важливість продовження розробки та впровадження передових методів кібербезпеки для забезпечення захисту сучасної інформаційної інфраструктури.

Попри значний прогрес у розробці та впровадженні технологій шифрування та цифрових підписів, існують специфічні виклики та обмеження, які все ще залишаються актуальними та потребують подальшого дослідження та вдосконалення. Розглянемо деякі з цих аспектів:

Масштабування та продуктивність: Однією з головних проблем, яка часто виникає при застосуванні шифрування в хмарних сервісах, є зниження продуктивності. Шифрування може спричинити затримки у відгуку системи, особливо при роботі з великими обсягами даних та користувачами. Це вимагає від розробників знаходження балансу між безпекою та продуктивністю.

Управління ключами: Керування криптографічними ключами є складним і часом ризикованим процесом, який включає генерацію, зберігання, обмін, використання та вилучення ключів. Неналежне управління ключами може призвести до витоку даних або навіть до втрати доступу до інформації.

Сумісність із законодавством: Забезпечення відповідності шифрувальних рішень законодавчим вимогам, таким як GDPR в Європі чи CCPA в Каліфорнії, є викликом, особливо в міжнародному та багатонаціональному контексті. Ці правила часто вимагають ретельного документування процесів шифрування та цифрових підписів, а також забезпечення прав на захист даних для кінцевих користувачів. Звичайно зараз розроблений додаток ще не є відповідним.

Технічні вразливості: Хоча шифрування є ефективним засобом захисту даних, самі криптографічні алгоритми та їх реалізації можуть містити вразливості, які можуть бути використані зловмисниками. Наприклад, помилки в програмному забезпеченні, що реалізує алгоритми шифрування, можуть зробити всю систему вразливою.

Ці виклики вимагають неперервних досліджень та розробки в області кібербезпеки, з метою вдосконалення існуючих технологій та створення нових рішень, які могли б забезпечити ефективний захист даних у хмарних сервісах без істотного впливу на їх доступність та продуктивність. Розглянуте дослідження підкреслює необхідність подальшого вивчення та розвитку криптографічних технологій, щоб вони могли ефективно справлятися із зростаючими вимогами сучасного цифрового світу.

На основі проведеного рішення, в якому було вивчено сучасні методи шифрування та цифрових підписів, можна визначити кілька ключових

рекомендацій та напрямків для подальшої роботи в цій галузі. Ці рекомендації мають на меті підвищення ефективності захисту даних та забезпечення кращої адаптації до мінливих умов кіберзагроз.

Розвиток нових криптографічних алгоритмів: Важливо продовжувати роботу над створенням і впровадженням нових криптографічних алгоритмів, які можуть ефективніше справлятися зі складністю сучасних кіберзагроз, особливо в контексті квантових обчислень, які можуть зробити багато з існуючих методів шифрування застарілими.

Покращення методів управління ключами: Розробка більш надійних та масштабованих систем управління криптографічними ключами є критично важливою для забезпечення безпеки даних на різних рівнях їх використання та зберігання.

Вдосконалення процедур відповідності та аудиту: Забезпечення дотримання міжнародних стандартів і нормативних вимог є ключовим для підтримки довіри користувачів і клієнтів. Необхідно вдосконалити процедури аудиту та відповідності для адаптації до нових вимог безпеки.

Інтеграція із розумними технологіями: Важливо інтегрувати криптографічні рішення із розумними технологіями, такими як ШІ і машинне навчання, для автоматизації процесів виявлення і реагування на загрози в реальному часі.

Підвищення обізнаності та навчання: Освіта і тренінги для співробітників у сфері кібербезпеки є необхідними для підвищення загального рівня захисту організацій. Важливо проводити регулярні навчальні сесії та тренінги з кібербезпеки.

Ці рекомендації мають на меті сприяти подальшому розвитку кібербезпеки у хмарних обчисленнях та підтримці високого рівня захисту даних у світі, де технології та загрози розвиваються швидкими темпами. Особлива увага повинна бути спрямована на інноваційні дослідження і вдосконалення технологічних рішень, здатних ефективно протистояти новим та майбутнім викликам у сфері кібербезпеки.

На отриманих результатів та аналізу існуючих методів захисту даних в хмарних обчисленнях, ми можемо розглянути шляхи практичного впровадження рекомендацій і зорієнтуватися на майбутні технологічні тренди, які формуватимуть сферу кібербезпеки.

Застосування гібридних криптографічних систем: З огляду на потребу у балансі між безпекою та продуктивністю, гібридні системи, які комбінують симетричні та асиметричні методи шифрування, можуть запропонувати оптимальне рішення. Це дозволить швидше шифрування великих обсягів даних з можливістю безпечного обміну ключами.

Інтеграція криптографії з блокчейн технологіями: Враховуючи потребу в підвищеній прозорості та відповідності до нормативних вимог, блокчейн може забезпечити не тільки безпеку даних, а й надійне логування всіх операцій, що полегшує аудит і забезпечує відповідність законодавству.

Розвиток квантово-стійких алгоритмів шифрування: З появою квантових обчислень стає очевидною потреба в алгоритмах, стійких до квантового декодування. Інвестиції в розробку та впровадження квантово-стійких криптографічних рішень стануть ключовими для захисту інформації в довгостроковій перспективі.

Адаптація до штучного інтелекту та машинного навчання: Використання ШІ для автоматизації процесів моніторингу та відповіді на загрози може значно підвищити ефективність систем кібербезпеки. ШІ може допомогти у виявленні незвичайних паттернів поведінки або потенційних вразливостей у режимі реального часу.

Формування міжгалузевих «наскрізних» стандартів безпеки: Розробка та прийняття уніфікованих стандартів, які б могли застосовуватися в різних галузях, допоможе створити стійку основу для захисту даних на глобальному рівні, зменшуючи конфлікти та непорозуміння між різними ринковими учасниками.

Реалізація цих рекомендацій потребуватиме координованих зусиль між розробниками, науковцями, бізнесом та державними регуляторами. Прогрес у

цих напрямках не лише підвищить рівень безпеки даних у хмарних обчисленнях, але й забезпечить готовність до викликів, які можуть виникнути в майбутньому.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. — National Institute of Standards and Technology, U.S. Department of Commerce. — 7 с. — (NIST Special Publication 800-145).
2. Leavitt, N. (2009). Is Cloud Computing Really Ready for Prime Time? — Computer, IEEE Computer Society. — 12 с.
3. Vogels, W. (2008). Amazon's Approach to Cloud Computing. — Amazon Web Services, LLC. — 8 с.
4. Marston, S. et al. (2011). Cloud Computing — The Business Perspective. — Decision Support Systems, Elsevier. — 35 с.
5. Zissis, D., & Lekkas, D. (2012). Addressing Cloud Computing Security Issues / D. Zissis, D. Lekkas. — Amsterdam : Future Generation Computer Systems, Elsevier, 2012. — 20 с.
6. Subashini, S., & Kavitha, V. (2011). A Survey on Security Issues in Service Delivery Models of Cloud Computing / S. Subashini, V. Kavitha. — London : Journal of Network and Computer Applications, Elsevier, 2011. — 24 с.
7. Stallings, W. (2005). Cryptography and Network Security: Principles and Practices / W. Stallings. — Upper Saddle River : Pearson Education, 2005. — 592 с.
8. Diffie, W., & Hellman, M. (1976). New Directions in Cryptography / W. Diffie, M. Hellman. — San Francisco : IEEE Transactions on Information Theory, 1976. — 12 с.
9. Kawushika, B. (2013). Adobe Cyberattack 2013 Case Study [Електронний ресурс]. Доступно на: https://www.linkedin.com/pulse/adobe-cyberattack-2013-case-study-bulitha-kawushika-hlrxc/
10. Proton Team. Dropbox Security Issues [Електронний ресурс]. Доступно на: https://proton.me/blog/dropbox-security-issues
11. Pagliery, J. (2017). Verizon data of 6 million users leaked online [Електронний ресурс]. CNN. Доступно на:

https://money.cnn.com/2017/07/12/technology/verizon-data-leaked-online/index.html

12. GeeksforGeeks. Security Issues in Cloud Computing [Електронний ресурс]. Доступно на: https://www.geeksforgeeks.org/security-issues-in-cloud-computing/
13. Veritis. Top 10 Security Issues in Cloud Computing [Електронний ресурс]. Доступно на: https://www.veritis.com/blog/top-10-security-issues-in-cloud-computing/
14. Wikipedia. Advanced Encryption Standard [Електронний ресурс]. Доступно на: https://uk.wikipedia.org/wiki/Advanced_Encryption_Standard
15. RSA. [Електронний ресурс]. Доступно на: https://www.rsa.com/
16. MySQL. [Електронний ресурс]. Доступно на: https://www.mysql.com/
17. Microsoft. Signature - .NET API [Електронний ресурс]. Доступно на:https://learn.microsoft.com/en-us/dotnet/api/java.security.signature?view=net-android-34.0
18. Oracle. Cipher - Java API [Електронний ресурс]. Доступно на: https://docs.oracle.com/javase/8/docs/api/javax/crypto/Cipher.html
19. Capital One. (2019). Facts 2019 [Електронний ресурс]. Доступно на: https://www.capitalone.com/digital/facts2019/
20. Репозіторій з кодом програми. [Електронний ресурс]. Доступно на: https://github.com/YashinAndrii/Kursova/tree/1-branch
21. Draw.io – утиліта для будування графіків [Електронний ресурс]. Доступно на: https://app.diagrams.net/

Додаток А

Лістинг А.1 – Клас EncryptionApp

```
public class EncryptionApp extends Application {


    private TextArea inputTextArea;
    private TextArea outputTextArea;
    private KeyStorage keyStorage;
    public static int userId;


    public static byte[] temp;


    public void startMainApplication(Stage primaryStage) throws
NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, IOException, InvalidKeyException {
        // Налаштування заголовка вікна
        primaryStage.setTitle("Шифрування і дешифрування");


        // Створення елементів управління
        Label inputLabel = new Label("Введіть текст:");
        inputTextArea = new TextArea();
        inputTextArea.setWrapText(true);
        inputTextArea.setPrefRowCount(5);


        Label outputLabel = new Label("Результат:");
        outputTextArea = new TextArea();
        outputTextArea.setWrapText(true);
        outputTextArea.setPrefRowCount(5);
        outputTextArea.setEditable(false);
        Button signButton = getSignButton();
```

```
        // Кнопка для перевірки підпису
        Button verifyButton = getVerifyButton();

        // Кнопка для переходу до панелі шифрування файлів
        Button encryptFileButton = getEncryptFileButton();

        // Кнопка для шифрування файлів
        Button encryptButton = generateButtonToEncrypt();

        // Кнопка для дешифрування файлів
        Button decryptButton = generateButtonToDecrypt();

        // Налаштування макету
        VBox layout = new VBox(10);
        layout.setPadding(new Insets(10));
        layout.getChildren().addAll(inputLabel, inputTextArea, outputLabel,
outputTextArea, encryptButton, decryptButton, encryptFileButton);
        layout.getChildren().addAll(signButton, verifyButton);

        // Відображення вікна
        primaryStage.setScene(new Scene(layout, 400, 300));
        primaryStage.show();
    }

    private Button getEncryptFileButton() {
        Button encryptFileButton = new Button("Шифрувати файл");
        encryptFileButton.setOnAction(e -> {
            FileEncryptionApp fileEncryptionApp = new FileEncryptionApp();
            Stage stage = new Stage();
            fileEncryptionApp.start(stage);
```

```
        });
        return encryptFileButton;
    }

    private Button getVerifyButton() {
        Button verifyButton = new Button("Перевірити підпис");
        verifyButton.setOnAction(e -> {
            try {
                String[] parts = outputTextArea.getText().split("\nПідпис: ");
                if (parts.length < 2) {
                    Dialog.showAlert("Помилка", "Недостатньо даних для перевірки підпису", Alert.AlertType.ERROR);
                    return;
                }
                String signature = parts[1];

                boolean isValid = keyStorage.verifySignature(parts[0].replace("Підписана інформація: ", ""), signature);
                Dialog.showAlert("Результат перевірки", "Підпис " + (isValid ? "вірна" : "невірна"), Alert.AlertType.INFORMATION);
            } catch (Exception ex) {
                Dialog.showAlert("Помилка", "Помилка при перевірці підпису: " + ex.getMessage(), Alert.AlertType.ERROR);
                ex.printStackTrace();
            }
        });
        return verifyButton;
    }

    private Button getSignButton() {
        Button signButton = new Button("Підписати");
```

```
        signButton.setOnAction(e -> {
            try {
                String data = inputTextArea.getText();
                KeyPair keyPair = CryptoUtils.generateKeyPair();
                byte[] signature = CryptoUtils.signData(data, keyPair.getPrivate());
                String signatureBase64 = Base64.getEncoder().encodeToString(signature);
                temp = signature;
                String publicKeyBase64 =
Base64.getEncoder().encodeToString(keyPair.getPublic().getEncoded());

                // Сохранение в базу данных
                keyStorage.saveSignature(data, signatureBase64, publicKeyBase64);

                outputTextArea.setText("Підписана інформація: " + data + "\nПідпис: " +
signatureBase64);
            } catch (Exception ex) {
                Dialog.showAlert("Помилка", "Помилка при створенні підпису: " +
ex.getMessage(), Alert.AlertType.ERROR);
                ex.printStackTrace();
            }
        });
        return signButton;
    }

    @Override
    public void start(Stage primaryStage) {
        // Налаштування заголовка вікна
        showLoginScreen(primaryStage);
    }

    public void showLoginScreen(Stage primaryStage) {
```

```
// Створення елементів форми логіну
GridPane grid = new GridPane();
grid.setAlignment(Pos.CENTER);
grid.setHgap(10);
grid.setVgap(10);
grid.setPadding(new Insets(25, 25, 25, 25));

keyStorage = new KeyStorage();

Text sceneTitle = new Text("Ласкаво просимо");
sceneTitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));
grid.add(sceneTitle, 0, 0, 2, 1);

Label userName = new Label("Ім'я користувача:");
grid.add(userName, 0, 1);

TextField userTextField = new TextField();
grid.add(userTextField, 1, 1);

Label pw = new Label("Пароль:");
grid.add(pw, 0, 2);

PasswordField pwBox = new PasswordField();
grid.add(pwBox, 1, 2);

Button btnLogin = new Button("Увійти");
Button btnRegister = new Button("Реєстрація");
HBox hbBtn = new HBox(10);
hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
hbBtn.getChildren().add(btnLogin);
```

```
        hbBtn.getChildren().add(btnRegister);
        grid.add(hbBtn, 1, 4);

        // Обробка подій кнопок
        btnLogin.setOnAction(e -> {
            if (keyStorage.authenticateUser(userTextField.getText(), pwBox.getText())) {
                primaryStage.hide();
                try {
                    startMainApplication(primaryStage);
                } catch (NoSuchPaddingException | IllegalBlockSizeException |
NoSuchAlgorithmException |
                         BadPaddingException | IOException | InvalidKeyException ex) {
                    throw new RuntimeException(ex);
                }
            } else {
                Dialog.showAlert("Помилка логіну", "Невірний логін або пароль!",
Alert.AlertType.ERROR);
            }
        });

        btnRegister.setOnAction(e -> showRegistrationForm());

        Scene scene = new Scene(grid, 300, 275);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void showRegistrationForm() {
        // Створення нового вікна для реєстрації
        Stage registerStage = new Stage();
        registerStage.setTitle("Реєстрація");
```

```
// Розмітка форми реєстрації
GridPane grid = new GridPane();
grid.setAlignment(Pos.CENTER);
grid.setHgap(10);
grid.setVgap(10);
grid.setPadding(new Insets(25, 25, 25, 25));

Label userNameLabel = new Label("Ім'я користувача:");
TextField userNameTextField = new TextField();
grid.add(userNameLabel, 0, 1);
grid.add(userNameTextField, 1, 1);

Label passwordLabel = new Label("Пароль:");
PasswordField passwordField = new PasswordField();
grid.add(passwordLabel, 0, 2);
grid.add(passwordField, 1, 2);

Button btnRegister = new Button("Зареєструватися");
HBox hbBtn = new HBox(10);
hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
hbBtn.getChildren().add(btnRegister);
grid.add(hbBtn, 1, 4);

// Обробник подій для кнопки реєстрації
btnRegister.setOnAction(e -> {

    boolean registered;
    try {
```

```
            registered = keyStorage.registerUser(userNameTextField.getText(),
passwordField.getText());
        } catch (NoSuchAlgorithmException ex) {
            throw new RuntimeException(ex);
        }
        if (registered) {
            Dialog.showAlert("Реєстрація", "Реєстрація успішна!",
Alert.AlertType.INFORMATION);
            registerStage.close();
        } else {
            Dialog.showAlert("Помилка реєстрації", "Не вдалося зареєструвати
користувача. Можливо, ім'я вже використовується.", Alert.AlertType.ERROR);
        }
    });

    // Підготовка і показ сцени
    Scene scene = new Scene(grid, 300, 275);
    registerStage.setScene(scene);
    registerStage.show();
}

//Кнопка шифрування
private Button generateButtonToEncrypt() {
    Button encryptButton = new Button("Шифрувати");
    encryptButton.setOnAction(e -> {
        try {
            String originalData = inputTextArea.getText();
            SecretKey chosenKey = chooseKeyDialog(); // Виклик діалогового вікна для
вибору ключа
            if (chosenKey != null) {
                String encryptedData = CryptoUtils.encryptText(originalData, chosenKey);
```

```
                outputTextArea.setText(encryptedData);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    });

    return encryptButton;
}

//Вибір використовувати існуючий ключ, чи згенерувати новий
private SecretKey chooseKeyDialog() {
    // Створення діалогового вікна для вибору ключа
    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
    alert.setTitle("Вибір ключа");
    alert.setHeaderText("Виберіть, який ключ використовувати:");
    alert.setContentText("Використати існуючий ключ, або згенерувати новий?");

    // Додавання кнопок "Використати існуючий ключ" та "Згенерувати новий ключ"
    ButtonType existingKeyButton = new ButtonType("Використати існуючий ключ");
    ButtonType generateKeyButton = new ButtonType("Згенерувати новий ключ");

    alert.getButtonTypes().setAll(existingKeyButton, generateKeyButton);

    Optional<ButtonType> result = alert.showAndWait();
    if (result.isPresent() && result.get() == existingKeyButton) {
        // Відкриття діалогового вікна для введення кодового слова
        TextInputDialog dialog = new TextInputDialog();
        dialog.setTitle("Введіть кодове слово");
```

```
        dialog.setHeaderText("Введіть кодове слово для використання існуючого ключа:");
        dialog.setContentText("Кодове слово:");

        Optional<String> inputSecretWord = dialog.showAndWait();
        if (inputSecretWord.isPresent()) {
            // Перевірка, чи введене кодове слово вірне
            String word = inputSecretWord.get();
            if (keyStorage.getSecretKeyByWord(word) != null) {
                //System.out.println(keyStorage.getSecretKeyByWord(word));// Повертаємо існуючий ключ
                return keyStorage.getSecretKeyByWord(word);
            } else {
                Dialog.showAlert("Помилка", "Невірне кодове слово!", Alert.AlertType.ERROR);
                return null; // Повертаємо null, якщо кодове слово невірне
            }
        } else {
            return null; // Повертаємо null, якщо користувач скасував введення кодового слова
        }
    } else {
        //Відкриття діалогового вікна для введення кодового слова
        TextInputDialog dialog = new TextInputDialog();
        dialog.setTitle("Введіть кодове слово");
        dialog.setHeaderText("Введіть кодове слово для генерації ключа:");
        dialog.setContentText("Кодове слово:");

        Optional<String> inputSecretWord = dialog.showAndWait();
        if (inputSecretWord.isPresent()) {
            // Перевірка, чи введене кодове слово вірне
```

```
            String secretWord = inputSecretWord.get();
            if (keyStorage.getSecretKeyByWord(secretWord) != null) {
                Dialog.showAlert("Помилка", "Таке слово вже існує",
Alert.AlertType.ERROR);
                return null;
            }
            // Повертаємо новий ключ
            SecretKey secretKey = CryptoUtils.secretKeyGen();
            keyStorage.saveKey(new Key(secretWord, secretKey));
            return secretKey;
        } else return null;
    }
}

private Button generateButtonToDecrypt() {
    Button decryptButton = new Button("Дешифрувати");
    decryptButton.setOnAction(e -> {
        try {
            String encryptedData = inputTextArea.getText();

            TextInputDialog dialog = new TextInputDialog();
            dialog.setTitle("Введіть кодове слово");
            dialog.setHeaderText("Введіть кодове слово для використання існуючого
ключа:");
            dialog.setContentText("Кодове слово:");

            Optional<String> inputSecretWord = dialog.showAndWait();
            if (inputSecretWord.isPresent()) {
                // Перевірка, чи введене кодове слово вірне
                String word = inputSecretWord.get();
                SecretKey key = keyStorage.getSecretKeyByWord(word);
```

```
                if (key != null) {
                    // Повертаємо існуючий ключ
                    String decryptedData = CryptoUtils.decryptText(encryptedData, key);
                    outputTextArea.setText(decryptedData);
                } else {
                    Dialog.showAlert("Помилка", "Невірне кодове слово!",
Alert.AlertType.ERROR);
                }
            }
        } catch (BadPaddingException ex) {
            // Обробка помилки при неправильному дешифруванні
            Dialog.showAlert("Помилка", "Помилка дешифрування: неправильні дані
або ключ.", Alert.AlertType.ERROR);
            ex.printStackTrace();
        } catch (Exception ex) {
            // Обробка інших винятків, які можуть виникнути під час дешифрування
            Dialog.showAlert("Помилка", "Помилка дешифрування: " +
ex.getMessage(), Alert.AlertType.ERROR);
            ex.printStackTrace();
        }
    });
    return decryptButton;
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```

Лістинг А.2 – Клас FileEncryptionApp

```
public class FileEncryptionApp extends Application {
```

```
private TextField filePathTextField;
private TextArea outputTextArea;
private KeyStorage keyStorage;

@Override
public void start(Stage primaryStage) {
    // Налаштування заголовка вікна
    primaryStage.setTitle("Шифрування файлів");

    keyStorage = new KeyStorage();

    // Створення елементів управління
    Label filePathLabel = new Label("Шлях до файлу:");
    filePathTextField = new TextField();
    filePathTextField.setPrefWidth(200);

    Button browseButton = new Button("Оберіть файл");
    browseButton.setOnAction(e -> browseFile());

    Button encryptButton = new Button("Зашифрувати");
    encryptButton.setOnAction(e -> encryptFile());

    Button decryptButton = new Button("Розшифрувати");
    decryptButton.setOnAction(e -> decryptFile());

    Button signButton = new Button("Підписати");
    signButton.setOnAction(e -> signFile());

    Button verifySignButton = new Button("Перевірити підпис");
    verifySignButton.setOnAction(e -> verifyFileSignature());
```

```
        Label outputLabel = new Label("Статус:");
        outputTextArea = new TextArea();
        outputTextArea.setPrefRowCount(3);
        outputTextArea.setEditable(false);

        // Налаштування макету
        VBox layout = new VBox(10);
        layout.setPadding(new Insets(10));
        layout.getChildren().addAll(filePathLabel, filePathTextField, browseButton,
encryptButton, decryptButton, signButton, verifySignButton, outputLabel,
outputTextArea);

        // Відображення вікна
        primaryStage.setScene(new Scene(layout, 400, 300));
        primaryStage.show();
    }

    private void browseFile() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Оберіть файл для шифрування");
        File selectedFile = fileChooser.showOpenDialog(null);
        if (selectedFile != null) {
            filePathTextField.setText(selectedFile.getAbsolutePath());
        }
    }

    private void encryptFile() {
        String filePath = filePathTextField.getText();
        File inputFile = new File(filePath);
        if (!inputFile.exists()) {
```

```
            outputTextArea.setText("Файл не знайдено.");
            return;
        }

        try {
            // Зашифрувати файл
            File encryptedFile = new File(inputFile.getParent(), "encrypted_" +
inputFile.getName());
            encryptFile(inputFile, encryptedFile);
            outputTextArea.setText("Файл зашифровано успішно.");
        } catch (Exception e) {
            outputTextArea.setText("Помилка під час шифрування файлу: " +
e.getMessage());
        }
    }

    private void decryptFile() {
        String filePath = filePathTextField.getText();
        File inputFile = new File(filePath);
        if (!inputFile.exists()) {
            outputTextArea.setText("Файл не знайдено.");
            return;
        }

        try {
            // Зашифрувати файл
            File decryptedFile = new File(inputFile.getParent(), "decrypted_" +
inputFile.getName());
            decryptFile(inputFile, decryptedFile);
            outputTextArea.setText("Файл розшифровано успішно.");
        } catch (Exception e) {
```

```
            outputTextArea.setText("Помилка під час розшифрування файлу: " +
e.getMessage());
        }
    }

    private void encryptFile(File inputFile, File outputFile) throws IOException,
NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException,
BadPaddingException, IllegalBlockSizeException {
        Cipher cipher = Cipher.getInstance("AES");
        SecretKey secretKey = chooseKeyDialog();
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);

        try (FileInputStream inputStream = new FileInputStream(inputFile);
             FileOutputStream outputStream = new FileOutputStream(outputFile)) {
            byte[] inputBytes = new byte[(int) inputFile.length()];
            inputStream.read(inputBytes);

            byte[] encryptedBytes = cipher.doFinal(inputBytes);
            outputStream.write(encryptedBytes);
        }
    }

    private void decryptFile(File inputFile, File outputFile) throws IOException,
NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException,
BadPaddingException, IllegalBlockSizeException {
        Cipher cipher = Cipher.getInstance("AES");
        TextInputDialog dialog = new TextInputDialog();
        dialog.setTitle("Введіть кодове слово");
        dialog.setHeaderText("Введіть кодове слово для використання існуючого
ключа:");
        dialog.setContentText("Кодове слово:");
```

```
        Optional<String> inputSecretWord = dialog.showAndWait();
        if (inputSecretWord.isPresent()) {
            // Перевірка, чи введене кодове слово вірне
            String word = inputSecretWord.get();
            if (keyStorage.getSecretKeyByWord(word) != null) {
                // Повертаємо існуючий ключ
                cipher.init(Cipher.DECRYPT_MODE,
keyStorage.getSecretKeyByWord(word));
            } else {
                Dialog.showAlert("Помилка", "Невірне кодове слово!",
Alert.AlertType.ERROR);
            }
        }

        try (FileInputStream inputStream = new FileInputStream(inputFile);
             FileOutputStream outputStream = new FileOutputStream(outputFile)) {
            byte[] inputBytes = new byte[(int) inputFile.length()];
            inputStream.read(inputBytes);

            byte[] decryptedBytes = cipher.doFinal(inputBytes);
            outputStream.write(decryptedBytes);
        }
    }

    private SecretKey chooseKeyDialog() {
        // Створення діалогового вікна для вибору ключа
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Вибір ключа");
        alert.setHeaderText("Виберіть, який ключ використовувати:");
        alert.setContentText("Використати існуючий ключ, або згенерувати новий?");
```

```
        // Додавання кнопок "Використати існуючий ключ" та "Згенерувати новий ключ"
        ButtonType existingKeyButton = new ButtonType("Використати існуючий ключ");
        ButtonType generateKeyButton = new ButtonType("Згенерувати новий ключ");


        alert.getButtonTypes().setAll(existingKeyButton, generateKeyButton);


        Optional<ButtonType> result = alert.showAndWait();
        if (result.isPresent() && result.get() == existingKeyButton) {
            // Відкриття діалогового вікна для введення кодового слова
            TextInputDialog dialog = new TextInputDialog();
            dialog.setTitle("Введіть кодове слово");
            dialog.setHeaderText("Введіть кодове слово для використання існуючого ключа:");
            dialog.setContentText("Кодове слово:");


            Optional<String> inputSecretWord = dialog.showAndWait();
            if (inputSecretWord.isPresent()) {
                // Перевірка, чи введене кодове слово вірне
                String word = inputSecretWord.get();
                if (keyStorage.getSecretKeyByWord(word) != null) {
                    return keyStorage.getSecretKeyByWord(word); // Повертаємо існуючий ключ
                } else {
                    Dialog.showAlert("Помилка", "Невірне кодове слово!", Alert.AlertType.ERROR);
                    return null; // Повертаємо null, якщо кодове слово невірне
                }
            } else {
                return null; // Повертаємо null, якщо користувач скасував введення кодового слова
```

```
            }
        } else {
            TextInputDialog dialog = new TextInputDialog();
            dialog.setTitle("Введіть кодове слово");
            dialog.setHeaderText("Введіть кодове слово для генерації ключа:");
            dialog.setContentText("Кодове слово:");

            Optional<String> inputSecretWord = dialog.showAndWait();
            if (inputSecretWord.isPresent()) {
                // Перевірка, чи введене кодове слово вірне
                String secretWord = inputSecretWord.get();
                if (keyStorage.getSecretKeyByWord(secretWord) != null) {
                    Dialog.showAlert("Помилка", "Таке слово вже існує",
Alert.AlertType.ERROR);
                    return null;
                }
                // Повертаємо новий ключ
                SecretKey secretKey = CryptoUtils.secretKeyGen();
                keyStorage.saveKey(new Key(secretWord, secretKey));
                return secretKey;
            } else return null;// Повертаємо новий ключ
        }
    }

    public void signFile() {
        try {
            // Чтение содержимого файла
            String filePath = filePathTextField.getText();
            File inputFile = new File(filePath);
            if (!inputFile.exists()) {
```

```
            outputTextArea.setText("Файл не знайдено.");
            return;
        }
        byte[] fileData = Files.readAllBytes(inputFile.toPath());

        // Получение приватного ключа пользователя
        KeyPair keyPair = CryptoUtils.generateKeyPair();
        byte[] signature = CryptoUtils.signData(fileData, keyPair.getPrivate());
        String signatureBase64 = Base64.getEncoder().encodeToString(signature);

        // Сохранение подписи в базе данных
        outputTextArea.setText("Підписана інформація: " + inputFile.getName() +
"\nПідпис: " + signatureBase64);
        keyStorage.saveFileSignature(inputFile.getName(), fileData, signatureBase64,
Base64.getEncoder().encodeToString(keyPair.getPublic().getEncoded()));
    } catch (Exception ex) {
        Dialog.showAlert("Помилка", "Помилка при створенні підпису: " +
ex.getMessage(), Alert.AlertType.ERROR);
        ex.printStackTrace();
    }
  }
  public void verifyFileSignature() {
    String[] parts = filePathTextField.getText().split(" Підпис: ");
    System.out.println(Arrays.toString(parts));
    if (parts.length < 2) {
        Dialog.showAlert("Помилка", "Недостатньо даних для перевірки підпису",
Alert.AlertType.ERROR);
        return;
    }
    String signature = parts[1];
    String filePath = parts[0].replace("Підписана інформація: ", "");
```

```
        File inputFile = new File(filePath);
        if (!inputFile.exists()) {
            outputTextArea.setText("Файл не знайдено.");
            return;
        }
        try {
            byte[] fileData = Files.readAllBytes(inputFile.toPath());
            boolean isValid = keyStorage.verifyFileSignature(fileData,
inputFile.getName(),signature);
            Dialog.showAlert("Результат перевірки", "Підпис " + (isValid ? "вірний" :
"невірний"), Alert.AlertType.INFORMATION);
        } catch (IOException e) {
            Dialog.showAlert("Результат перевірки", "Підпис невірний",
Alert.AlertType.INFORMATION);
            throw new RuntimeException(e);
        } catch (IllegalArgumentException e) {
            Dialog.showAlert("Результат перевірки", "Підпис невірний",
Alert.AlertType.INFORMATION);
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Лістинг А.3 – Клас CryptoUtils

```
public class CryptoUtils {
    public static KeyPair generateKeyPair() throws NoSuchAlgorithmException {
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(2048);
```

```
        return keyGen.generateKeyPair();
    }

    public static String hashPassword(String password, String salt) throws
NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        // Додаємо сіль до пароля перед хешуванням
        String saltedPassword = salt + password;
        byte[] hashedBytes = md.digest(saltedPassword.getBytes());
        return Base64.getEncoder().encodeToString(hashedBytes);
    }

    public static String generateSalt(int length) {
        SecureRandom random = new SecureRandom();
        byte[] salt = new byte[length];
        random.nextBytes(salt);
        return Base64.getEncoder().encodeToString(salt);
    }

    // Метод для создания подписи
    public static byte[] signData(String data, PrivateKey privateKey) throws Exception {
        Signature signature = Signature.getInstance("SHA256withRSA");
        signature.initSign(privateKey);
        signature.update(data.getBytes());
        return signature.sign();
    }

    public static byte[] signData(byte[] data, PrivateKey privateKey) throws Exception {
        Signature signature = Signature.getInstance("SHA256withRSA");
        signature.initSign(privateKey);
```

```
        signature.update(data);
        return signature.sign();
    }

    public static SecretKey secretKeyGen() {
        // Генерація випадкового секретного ключа
        SecureRandom secureRandom = new SecureRandom();
        byte[] keyBytes = new byte[16]; // 16 байт для ключа AES-128
        secureRandom.nextBytes(keyBytes);
        SecretKeySpec secretKeySpec = new SecretKeySpec(keyBytes, "AES");

        // Перетворення SecretKeySpec в SecretKey
        return new SecretKeySpec(secretKeySpec.getEncoded(), "AES");
    }

    public static String encryptText(String data, SecretKey secretKey) throws
NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeyException,
IllegalBlockSizeException, BadPaddingException {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedData = cipher.doFinal(data.getBytes());
        return Base64.getEncoder().encodeToString(encryptedData);
    }

    public static String decryptText(String encryptedData, SecretKey secretKey) throws
BadPaddingException {
        try {
            Cipher cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            byte[] decryptedData =
cipher.doFinal(Base64.getDecoder().decode(encryptedData));
```

```
            return new String(decryptedData);
        } catch (BadPaddingException e) {
            // Обробка помилки при неправильному дешифруванні
            System.err.println("Помилка дешифрування: неправильні дані або ключ.");
            e.printStackTrace();
            Dialog.showAlert("Помилка", "Невірне кодове слово!",
Alert.AlertType.ERROR);
            return null;
        } catch (Exception e) {
            // Обробка інших винятків, які можуть виникнути під час дешифрування
            System.err.println("Помилка дешифрування: " + e.getMessage());
            e.printStackTrace();
            return null;
        }
    }

    public static String hashText(String password) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hashedBytes = md.digest(password.getBytes(StandardCharsets.UTF_8));
            return Base64.getEncoder().encodeToString(hashedBytes);
        } catch (NoSuchAlgorithmException ex) {
            throw new RuntimeException("Не вдалося знайти алгоритм хешування", ex);
        }
    }
}
```

Лістинг А.4 – Клас Dialog

```
public class Dialog {
    public static void showAlert(String title, String message, Alert.AlertType alertType) {
```

```
        Alert alert = new Alert(alertType);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }
}
```

Лістинг А.5 – Клас Key

```
public class Key {

    private final String secretWord;
    private final SecretKey secretKey;

    public Key(String secretWord, SecretKey secretKey) {
        this.secretWord = secretWord;
        this.secretKey = secretKey;
    }

    public String getSecretWord() {
        return CryptoUtils.hashText(secretWord);
    }

    public SecretKey getSecretKey() {
        return secretKey;
    }
}
```

Лістинг А.6 – Клас KeyStorage

```
public class KeyStorage {

    // З'єднання з базою даних
    private Connection connect() throws SQLException {
        String url = "jdbc:mysql://localhost:3306/test";
        String user = "root";
        String password = "16122002aY";
        return DriverManager.getConnection(url, user, password);
    }

    public void saveKey(Key key) {
        String sql = "INSERT INTO `keys` (key_data, secret_word, user_id) VALUES (?,
?, ?)";
        try (Connection conn = connect();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setBytes(1, key.getSecretKey().getEncoded());
            pstmt.setString(2, key.getSecretWord());
            pstmt.setInt(3, EncryptionApp.userId);
            pstmt.executeUpdate();
            System.out.println("Key saved successfully!");
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }

    public boolean authenticateUser(String username, String password) {
        String sql = "SELECT password, salt FROM Users WHERE username = ?";

        try (Connection conn = this.connect();
```

```
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, username);

            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                String storedHash = rs.getString("password");
                String salt = rs.getString("salt");
                String hashedPassword = CryptoUtils.hashPassword(password, salt); // Хешування введеного пароля з використанням збереженої солі

                if (hashedPassword.equals(storedHash)) {
                    // Користувач аутентифікований
                    return true;
                }
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
        return false;
    }

    public boolean registerUser(String username, String password) throws NoSuchAlgorithmException {
        String sql = "INSERT INTO Users (username, password, salt) VALUES (?, ?, ?)";
        String salt = CryptoUtils.generateSalt(16); // Генерує випадкову сіль
        String hashedPassword = CryptoUtils.hashPassword(password, salt);

        try (Connection conn = this.connect();
```

```
            PreparedStatement pstmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {
            pstmt.setString(1, username);
            pstmt.setString(2, hashedPassword);
            pstmt.setString(3, salt);
            int affectedRows = pstmt.executeUpdate();

            if (affectedRows > 0) {
                try (ResultSet generatedKeys = pstmt.getGeneratedKeys()) {
                    if (generatedKeys.next()) {
                        EncryptionApp.userId = generatedKeys.getInt(1);
                        return true;
                    }
                }
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return false;
    }

    // Метод для отримання секретного ключа з бази даних
    public SecretKey getSecretKeyByWord(String word) {
        String sql = "SELECT key_data FROM `keys` WHERE secret_word = ?";
        try (Connection conn = connect();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, CryptoUtils.hashText(word)); // Параметр secret_word
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                byte[] keyBytes = rs.getBytes("key_data");
```

```
            return new SecretKeySpec(keyBytes, "AES");
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    return null;
}

public void saveSignature(String data, String signatureBase64, String publicKeyBase64) {
    String sql = "INSERT INTO Documents(user_id, document, signature, public_key) VALUES (?, ?, ?, ?)";
    try (Connection conn = this.connect();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, EncryptionApp.userId); // Текущий авторизованный пользователь
        pstmt.setString(2, data);
        pstmt.setString(3, signatureBase64);
        pstmt.setString(4, publicKeyBase64);
        pstmt.executeUpdate();

        try (ResultSet generatedKeys = pstmt.getGeneratedKeys()) {
            if (generatedKeys.next()) {
                generatedKeys.getInt(1);
            }
        }

    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
```

```
}

public boolean verifySignature(String data, String signatureBase64) {
    try {
        // Преобразуем подпись из Base64 в байты
        byte[] signatureBytes = Base64.getDecoder().decode(signatureBase64);
        System.out.println(Arrays.toString(signatureBytes));

        // Получаем публичный ключ пользователя из базы данных
        PublicKey publicKey = getPublicKey(data);
        if (publicKey == null) {
            return false;
        }
        System.out.println(publicKey);

        // Создаем объект Signature для проверки
        Signature sig = Signature.getInstance("SHA256withRSA");
        sig.initVerify(publicKey);
        sig.update(data.getBytes());

        // Проверяем подпись
        return sig.verify(signatureBytes);
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

public PublicKey getPublicKey(String data) {
```

```
        String sql = "SELECT public_key FROM Documents WHERE user_id = ? AND id
= (SELECT MAX(id) FROM Documents WHERE user_id = ? AND document = ?)";
        try (Connection conn = connect();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, EncryptionApp.userId);
            pstmt.setInt(2, EncryptionApp.userId);
            pstmt.setString(3, data);

            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    String publicKeyBase64 = rs.getString("public_key");
                    byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyBase64);
                    X509EncodedKeySpec keySpec = new
X509EncodedKeySpec(publicKeyBytes);
                    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
                    return keyFactory.generatePublic(keySpec);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    // Преобразует массив байтов в объект PublicKey
    public static PublicKey getPublicKeyFromBytes(byte[] publicKeyBytes) {
        try {
            X509EncodedKeySpec keySpec = new X509EncodedKeySpec(publicKeyBytes);
            KeyFactory keyFactory = KeyFactory.getInstance("RSA");
            return keyFactory.generatePublic(keySpec);
        } catch (Exception e) {
```

```
            e.printStackTrace();
            return null;
        }
    }

    public void saveFileSignature(String fileName, byte[] fileData, String
signatureBase64, String publicKeyBase64) {
        String sql = "INSERT INTO FileSignatures(user_id, file_name, data_hash,
signature, public_key) VALUES (?, ?, ?, ?, ?)";


        try (Connection conn = connect();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, EncryptionApp.userId);
            pstmt.setString(2, fileName);

            // Генерируем хэш для файла
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hash = digest.digest(fileData);
            String dataHashBase64 = Base64.getEncoder().encodeToString(hash);
            pstmt.setString(3, dataHashBase64);

            pstmt.setString(4, signatureBase64);
            pstmt.setString(5, publicKeyBase64);
            pstmt.executeUpdate();
        } catch (SQLException | NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
```

```
    public boolean verifyFileSignature(byte[] data, String fileName, String
signatureBase64) {
        String sql = "SELECT data_hash, signature, public_key FROM FileSignatures
WHERE user_id = ? AND id = (SELECT MAX(id) FROM FileSignatures WHERE
user_id = ? AND file_name = ?)";

        try (Connection conn = connect();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, EncryptionApp.userId);
            pstmt.setInt(2, EncryptionApp.userId);
            pstmt.setString(3, fileName);

            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    String dataHashBase64 = rs.getString("data_hash");
                    System.out.println(rs.getString("signature").equals(signatureBase64));
                    String publicKeyBase64 = rs.getString("public_key");
                    MessageDigest digest = MessageDigest.getInstance("SHA-256");
                    byte[] hash = digest.digest(data);

                    byte[] dataHash = Base64.getDecoder().decode(dataHashBase64);
                    if(!Arrays.equals(dataHash, hash)) {
                        Dialog.showAlert("Дані спотворено", "Дані спотворено!",
Alert.AlertType.ERROR);
                        return false;
                    }
                    byte[] signature = Base64.getDecoder().decode(signatureBase64);
                    byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyBase64);

                    PublicKey publicKey = getPublicKeyFromBytes(publicKeyBytes);
```

```
                return verifyDigitalSignature(data, signature, publicKey);
            }
        }
    } catch (SQLException | NoSuchAlgorithmException | InvalidKeyException |
            SignatureException e) {
        e.printStackTrace();
    }
    return false;
}

private boolean verifyDigitalSignature(byte[] data, byte[] signature, PublicKey
publicKey) throws NoSuchAlgorithmException, InvalidKeyException,
SignatureException {
    Signature sig = Signature.getInstance("SHA256withRSA");
    sig.initVerify(publicKey);
    sig.update(data);
    return sig.verify(signature);
}
}
```