пам’яті, виконання складних запитів, ризик втрати даних під час вимкнення системи. Ці обмеження важливо враховувати під час вибору Redis для конкретного проєкту та вживати відповідних заходів для оптимізації та захисту від потенційних проблем.

Список використаних джерел

1. Redis. URL: https://en.wikipedia.org/wiki/Redis (дата звернення: 20.11.2023).

2. Redis на практичних прикладах. URL: https://devzone.org.ua/post/redis-na-praktichnikh-prikladakh (дата звернення: 22.11.2023).

3. Introduction to Redis. URL: https://redis.io/docs/about/ (дата звернення: 22.11.2023).

УДК 004.65:004.056.5

Шевцов М. В., здобувач 2 курсу спеціальності 122 Комп’ютерні науки,
Гончар В. М., асистент кафедри інформаційних технологій

ВИКОРИСТАННЯ КРИПТОГРАФІЇ ДЛЯ ЗАХИСТУ ДАНИХ У БАЗІ ДАНИХ

Донецький національний університет імені Василя Стуса, м. Вінниця

У сучасному світі, коли інформація є найціннішим ресурсом, забезпечення безпеки даних стає критично важливим завданням для багатьох організацій. Паролі користувача, цифрові підписи та інша конфіденційна інформація може бути злита в мережу за будь-яких обставин – протиправних дій зловмисників чи через просту несправність сервера. Тому зберігати цю інформацію в базах даних треба у зашифрованому форматі. До того ж необхідно, щоб із шифру не можна було отримати ключ у тому вигляді, в якому його вводив користувач. Саме інтеграція криптографічних методів та хеш-функцій стає ключовим аспектом забезпечення конфіденційності та цілісності даних.

Найвідомішими криптографічними алгоритмами для захисту інформації, які використовуються зокрема і в базах даних, є:

- AES (Advanced Encryption Standard) – це симетричний блоковий метод, який використовує ключ для шифрування та розшифрування даних в блоках фіксованого розміру. Він забезпечує доволі високий рівень безпеки і широко використовується для шифрування даних у реляційних базах даних [1].
- RSA (Rivest–Shamir–Adleman) – це асиметричний алгоритм, який використовує пару ключів для шифрування та розшифрування даних. Публічний ключ

використовується для шифрування, а приватний – для дешифрування. Він відіграє важливу роль у захисті даних під час їх передачі та обміну [2].

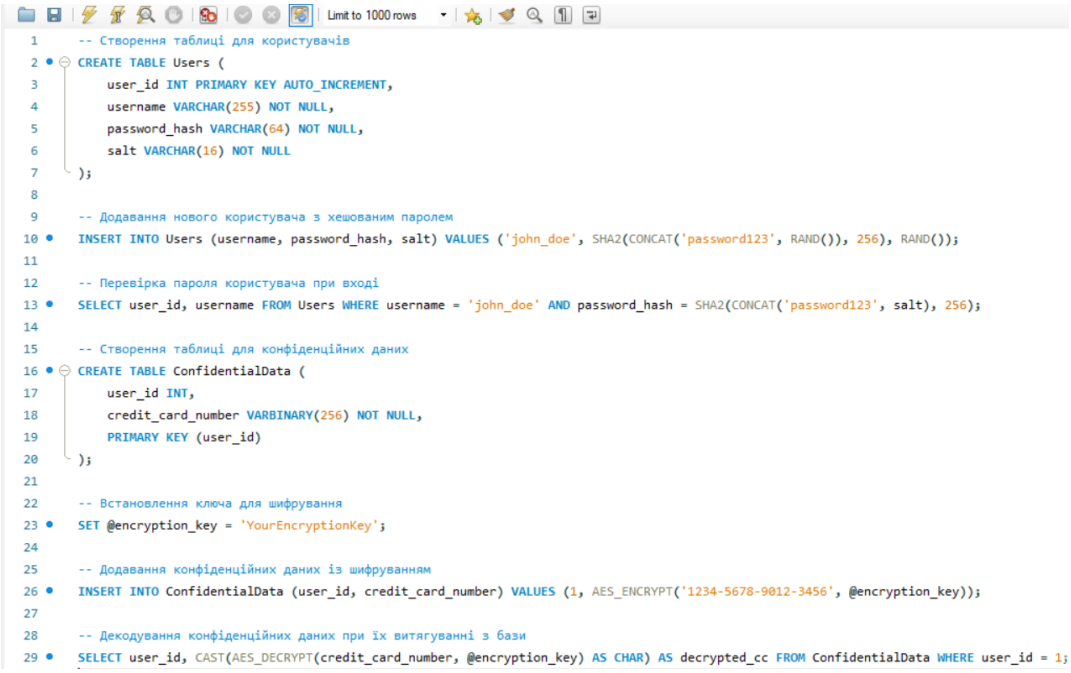
➢ SHA-256 (Secure Hash Algorithm 256-bit) – це хеш-функція, яка є однією з сімейства алгоритмів SHA. Вони відрізняються розміром вхідного хешу, стійкістю до колізій та швидкістю обчислення. Саме SHA-256 використовується для створення фіксованого 256-бітного хеш-коду зі вхідних даних. Вона застосовується для перевірки цілісності даних та генерації контрольних сум.

➢ MD5 (Message Digest Algorithm 5) – це хеш-функція, яка обчислює контрольну суму вхідного повідомлення. Результат – фіксована 128-бітна хеш-сума [2]. Хоча MD5 була широко використовуваною раніше, зараз вона не є безпечною і відома своєю вразливістю до колізій. Проте вона все ще використовується для простих завдань, як-от генерація хеш-сум файлів.

Для забезпечення більшого рівня безпеки існують концепції генерації одноразових ключів та використання солів. Генератори одноразових ключів – це алгоритми генерації унікальних ключів для кожної транзакції або сесії, які надають додатковий рівень безпеки, оскільки ключі використовуються лише один раз і не можуть бути використані для наступних атак. Один із таких алгоритмів – це TOTP (Time-Based One-Time Password), який базується на поточному часі та секретному ключі. Концепція солів – випадкових рядків, які додаються до паролів користувачів перед хешуванням, – підвищує безпеку, оскільки два однакові паролі будуть мати різні хеш-значення через випадковий символ, який додається до кожного пароля. Солі важливі для захисту бази даних від атак методом перебору (brute force) та атак методом таблиць радужних хешів (rainbow tables) [3].

Спробуємо розібратись на практиці. Розглянемо приклад шифрування конфіденційних даних (наприклад, номера кредитної картки).

Тут ми створюємо таблицю Users, яка містить інформацію про користувачів, як-от ідентифікатор (user_id), ім’я користувача (username), хеш пароля (password_hash) і випадковий рядок (salt). Код вставляє нового користувача в таблицю Users зі згенерованим хешем пароля і випадковим рядком солі для безпеки пароля. Під час входу програма виконує запит до таблиці Users, де перевіряє, чи співпадає хеш введеного пароля користувача з тим, що зберігається в базі даних. Таблиця ConfidentialData призначена для зберігання конфіденційних даних, як-от номери кредитних карток. Для цієї таблиці використовується VARBINARY для зберігання зашифрованих даних. Щоб забезпечити безпеку конфіденційних даних, встановлюється ключ шифрування (@encryption_key), який використовується для шифрування та розшифрування даних. Код вставляє новий запис у таблицю ConfidentialData, де номер кредитної картки зашифрований з використанням ключа шифрування. Під час витягування конфіденційних даних із таблиці ConfidentialData код розшифровує зашифрований номер кредитної карти за допомогою ключа шифрування.



```sql
-- Створення таблиці для користувачів
CREATE TABLE Users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255) NOT NULL,
    password_hash VARCHAR(64) NOT NULL,
    salt VARCHAR(16) NOT NULL
);

-- Додавання нового користувача з хешованим паролем
INSERT INTO Users (username, password_hash, salt) VALUES ('john_doe', SHA2(CONCAT('password123', RAND()), 256), RAND());

-- Перевірка пароля користувача при вході
SELECT user_id, username FROM Users WHERE username = 'john_doe' AND password_hash = SHA2(CONCAT('password123', salt), 256);

-- Створення таблиці для конфіденційних даних
CREATE TABLE ConfidentialData (
    user_id INT,
    credit_card_number VARBINARY(256) NOT NULL,
    PRIMARY KEY (user_id)
);

-- Встановлення ключа для шифрування
SET @encryption_key = 'YourEncryptionKey';

-- Додавання конфіденційних даних із шифруванням
INSERT INTO ConfidentialData (user_id, credit_card_number) VALUES (1, AES_ENCRYPT('1234-5678-9012-3456', @encryption_key));

-- Декодування конфіденційних даних при їх витягуванні з бази
SELECT user_id, CAST(AES_DECRYPT(credit_card_number, @encryption_key) AS CHAR) AS decrypted_cc FROM ConfidentialData WHERE user_id = 1;
```

Рисунок 1. Приклад бази даних

Ми використали функцію SHA-2 з сімейства SHA для створення хешу паролів, додавали сіль (salt) для підвищення безпеки, і використали функції AES_ENCRYPT() та AES_DECRYPT(), які працюють за згаданим раніше алгоритмом AES, для шифровки та дешифровки конфіденційних даних у MySQL.

Отже, інтеграція криптографічних методів та хеш-функцій у бази даних є основним фактором забезпечення безпеки інформації. Криптографічні методи дають змогу захистити дані від несанкціонованого доступу та перегляду, а хеш-функції забезпечують надійне шифрування й перевірку цілісності даних. Інтеграція цих методів дає змогу створити надійну систему зберігання та обробки даних, що важливо для багатьох сфер діяльності.

Список використаних джерел

1. Database Encryption in SQL Server 2008 Enterprise Edition. URL: https://download.microsoft.com/download/8/b/2/8b22991f-3f2f-4cea-b2ba-55c190841145/tdeandefsbitlocker.docx

2. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley, 1996. 784 c.

3. Давлетханов М. Концепция одноразовых паролей в построении системы аутентификации Byte, 2006. Vol. 7–8(95).