Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки
та захисту інформації
Іван ПАРХОМЕНКО
«___» ________ 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *125 Кібербезпека та захист інформації*
(код і назва спеціальності)

освітній ступень *магістр*

освітньо-наукова програма *Кібербезпека*
(назва освітньої програми)

на тему: «Метод захисту та забезпечення стійкості паролів у базах даних веб-застосунків»

Виконавець: студент II курсу, групи КБм-21

________________ **Павло ГОРБАТЮК**
(підпис) (Ім'я, ПРІЗВИЩЕ)

	Ім'я, ПРІЗВИЩЕ	**Підпис**
Науковий керівник	Іван ПАРХОМЕНКО	
Нормоконтроль	Сергій ДАКОВ	

Київ 2025

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки
та захисту інформації
Іван ПАРХОМЕНКО
«25» жовтня 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека та захист інформації*
(код і назва спеціальності)

освітній ступень *магістр*

Здобувача КБм-21 Горбатюка Павла Володимировича
(група) (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Метод захисту та забезпечення стійкості паролів у базах даних веб-застосунків

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 4 від 24.10.2024 р.

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень Процес надання та забезпечення цілісності зберігання паролів у базах-даних веб-застосунків

Предмет досліджень Методи захисту паролів від можливості їх отримання зловмисниками, а також методи отримання або зламу паролів користувачів веб-застосунків

Мета Розробка методу захисту та забезпечення стійкості паролів у веб-застосунках

Вихідні дані для проведення роботи Методи захисту паролів у базах даних веб-застосунків

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна створення нового методу збереження цілісності паролів та їх захисту від зловмисників у базах даних веб-застосунків, з метою запобіганню їх розкриття та створення безпечних умов зберігання

Практична цінність застосування розробленого методу розробниками веб-застосунків для підвищення рівня захисту даних автентифікації користувачів

4. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Уточнення постановки задачі	25.10.2024 – 05.11.2024
Аналіз літературних джерел	10.12.2025 – 13.01.2025
Розгляд нормативно-правової та законодавчої бази, що регулює питання збереження даних користувачів	15.01.2025 – 17.01.2025
Аналіз вразливостей та загроз при зберіганні паролів у базах даних	18.01.2025 – 23.01.2025
Дослідження сучасних методів зберігання паролів	25.01.2025 – 28.01.2025
Виокремлення основних принципів для захисту паролів	08.02.2025 – 11.02.2025
Розробка нового методу зберігання паролів та імплементація його до веб-застосунку	15.02.2025 – 25.03.2025
Опис розробленого методу	01.04.2025 – 10.04.2025
Виконання порівняння розробленого методу з іншими сучасним аналогами для збереження чутливих даних	15.04.2025 – 29.04.2025
Оформлення пояснювальної записки згідно методичних рекомендацій	05.05.2025 – 15.05.2025
Подача пакету документів на розгляд ЕК	15.05.2025 – 19.05.2025

Завдання видав ____________ Іван ПАРХОМЕНКО
(підпис) (Ім'я, ПРІЗВИЩЕ)

Завдання прийняв
до виконання ____________ Павло ГОРБАТЮК
(підпис) (Ім'я, ПРІЗВИЩЕ)

Дата видачі завдання: 25.10.2024 р.
Термін подання кваліфікаційної роботи до ЕК 19.05.2025 р.

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Метод захисту та забезпечення стійкості паролів у базах даних веб-застосунків»: 94 сторінки, 49 рисунків, 6 таблиць, 49 літературних джерел.

Метою роботи є розробка методу захисту та забезпечення стійкості паролів у веб-застосунках.

Об'єктом дослідження є процес надання та забезпечення цілісності зберігання паролів у базах-даних веб-застосунків.

Предметом дослідження є методи захисту паролів від можливості їх отримання зловмисниками, а також методи отримання або зламу паролів користувачів веб-застосунків.

Наукова новизна полягає у створенні нового методу збереження цілісності паролів та їх захисту від зловмисників у базах даних веб-застосунків, з метою запобіганню їх розкриття та створення безпечних умов зберігання.

Практична цінність полягає у можливості застосування розробленого методу розробниками веб-застосунків для підвищення рівня захисту даних автентифікації користувачів.

Методи дослідження:

- Аналіз науково-технічної літератури.
- Аналіз сучасних загроз та вразливостей паролів.
- Моделювання архітектури веб-застосунку з реалізованим в ньому методі.
- Порівняння реалізованого методу з іншими сучасними аналогами.

Ключові слова: хешування, шифрування, веб-застосунок, база даних, стійкість до компрометації, захист даних.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

CIA – Конфіденційність, цілісність, доступність

GDPR – General Data Protection Regulation

ХФ – Хеш-функція

AES – Advanced Encryption Standard

SHA – Secure Hash Algorithms

IV – Вектор ініціалізації

NIST – National Institute of Standards and Technology

API – Application Programming Interface

ЗМІСТ

ЗМІСТ 6

ВСТУП.. 8

РОЗДІЛ 1 ОГЛЯД НОРМАТИВНИХ ВИМОГ ТА ТИПОВИХ ПІДХОДІВ ДО ЗБЕРІГАННЯ ПАРОЛІВ 10

1.1 Нормативно-правова база та стандарти безпеки 10

1.2 Типові загрози та вразливості при використанні паролів 14

1.2.1 Основні вразливості паролів 14

1.2.2 Загрози властиві при використанні паролів 19

1.3 Класифікація сучасних методів зберігання паролів. Їх переваги та недоліки 25

1.3.1 Сучасні методи зберігання паролів у базах даних 25

1.3.2 Переваги та недоліки обраних методів зберігання паролів 29

1.4 Висновки до першого розділу 31

РОЗДІЛ 2 АНАЛІЗ АЛГОРИТМІВ ЗАХИСТУ ДЛЯ ЗБЕРІГАННЯ ПАРОЛІВ.. 32

2.1 Ключові принципи у захисті паролів 32

2.1.1 Загальні технічні особливості збереження паролів 32

2.1.2 Важливість окремого зберігання ключів при шифрування 34

2.1.3 Роль векторів ініціалізації при симетричному шифруванні 35

2.2 Огляд алгоритмів AES-256, SHA-512 та bcrypt...................................... 36

2.2.1 Алгоритм SHA-512 36

2.2.2 Алгоритм шифрування AES... 44

2.2.3 Алгоритм Bcrypt... 55

2.3 Обґрунтування вибору алгоритмів для розробленого методу 58

2.4 Висновки до другого розділу 61

РОЗДІЛ 3 РОЗРОБКА МЕТОДУ ЗБЕРІГАННЯ ПАРОЛІВ У ВЕБ-ЗАСТОСУНКАХ.. 62

3.1 Засоби реалізації розробленого веб-застосунку.. 62

3.2 Реалізації алгоритму шифрування та хешування...................................... 68

3.3 Опис додаткових функціональних компонентів веб-додатку 72

3.4 Висновки до третього розділу... 74

РОЗДІЛ 4 ДЕМОНСТРАЦІЯ МОЖЛИВОСТЕЙ РОЗРОБЛЕНОГО РІШЕННЯ 75

4.1 Порівняння існуючих рішень з розробленим методом 75

4.1.1 Навантаження на систему та час виконання 76

4.1.2 Порівняння захищеності методів 80

4.2 Висновки четвертого до розділу.. 86

ВИСНОВКИ.. 88

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 90

ДОДАТОК А.. 95

ДОДАТОК Б 99

ВСТУП

В епоху стрімкого розвитку цифрових технологій та повсюдного використання веб-застосунків, забезпечення надійного захисту користувацьких даних стає одним із найважливіших пріоритетів. Особливу увагу потрібно приділяти захисту для даних автентифікації, зокрема паролів, оскільки їх компрометація може призвести до несанкціонованого доступу до конфіденційної інформації, фінансових втрат та репутаційної шкоди як для користувачів, так і для організацій. Частота та складність кібератак, спрямованих на викрадення облікових даних, невпинно зростають, що вимагає постійного вдосконалення існуючих та розробки нових, більш стійких механізмів захисту.

Традиційні підходи до зберігання паролів у базах даних, такі як зберігання у відкритому вигляді або використання простих хеш-функцій без додаткових заходів безпеки (наприклад, солі), давно визнані небезпечними та вразливими до різноманітних атак, включаючи атаки за словником, повного перебору та атак з використанням райдужних таблиць. Сучасні стандарти рекомендують використання адаптивних криптографічних хеш-функцій, таких як bcrypt, scrypt або Argon2, які спеціально розроблені для ускладнення масового перебору паролів завдяки високим вимогам до обчислювальних ресурсів та пам'яті, а також використанню солі для унікальності кожного хешу.

Незважаючи на ефективність цих методів, пошук шляхів подальшого підвищення рівня безпеки залишається актуальним завданням. Потенційні вразливості можуть виникати не лише на рівні самого алгоритму хешування, але й у контексті загальної архітектури системи безпеки, можливих атак на інфраструктуру зберігання даних чи витоків самих хешів. Це стимулює дослідження комбінованих підходів, які могли б забезпечити додатковий рівень захисту та стійкості до різних векторів атак.

Актуальність даної роботи зумовлена необхідністю розробки більш надійних методів захисту паролів у базах даних веб-застосунків, здатних

протистояти сучасним загрозам. Існуючі рішення, хоча й ефективні, постійно піддаються аналізу з боку зловмисників, що потребує проактивних дій у сфері кібербезпеки.

Метою роботи є розробка методу захисту та забезпечення стійкості паролів у веб-застосунках.

Для досягнення зазначеної мети кваліфікаційної роботи поставлено наступні **завдання**:

- Проаналізувати можливі сучасні загрози та вразливості при використанні та збереженні паролів.
- Визначити сучасні методи захисту паролів у базах даних.
- Розробити метод та імплементувати його у веб-застосунок.
- Виконати порівняння з іншими методами збереження чутливих даних.

Апробація роботи:

Горбатюк П., Пархоменко І. Метод захисту паролів у базах даних веб-застосунків. VIII Міжнародна науково-практична конференція «Проблеми кібербезпеки інформаційно-комунікаційних систем» (PCSICS). 2025. С. 175-176.

РОЗДІЛ 1
ОГЛЯД НОРМАТИВНИХ ВИМОГ ТА ТИПОВИХ ПІДХОДІВ ДО ЗБЕРІГАННЯ ПАРОЛІВ

1.1 Нормативно-правова база та стандарти безпеки

Законодавчі вимоги для безпечного зберігання паролів ґрунтується на кількох основоположних принципах, які широко закріплені в законах про захист даних у різних юрисдикціях. Ці принципи зазвичай сформульовані в загальних рисах, але забезпечують правове обґрунтування конкретних технічних вимог безпеки:

Перший принцип – мінімізація даних. Згідно з цим принципом, який є основним у сучасній юриспруденції щодо захисту даних, організації повинні збирати та зберігати лише те, що є абсолютно необхідним для законної мети. Цей принцип, коли йдеться про зберігання паролів, забороняє зберігати будь-яку непотрібну інформацію про облікові дані користувачів (наприклад, надмірно описові підказки до паролів). Крім того, він забороняє постійне зберігання паролів у відкритому вигляді після завершення початкової автентифікації.

Другий – безпека, а саме конфіденційність, цілісність, доступність. Вона є багатогранним принципом, який часто називають «тріадою CIA», який встановлює, що тримачі даних зобов’язані впроваджувати відповідні технічні та організаційні заходи для забезпечення безпеки всіх оброблюваних персональних даних. Цей принцип застосовується саме для дотримання вимог принципів тріади CIA:

- Конфіденційність: Організації зобов’язані перешкоджати несанкціонованому доступу до паролів, які вони зберігають у себе. Тому, для зменшення ймовірність витоку даних, є необхідним вжити надійний контроль доступу, шифрування (як під час передачі, так і під час зберігання) та

комплексних заходів безпеки системи, спрямованих на зниження ризику витоку даних.

- Цілість: Правові вимоги щодо збереження цілісності даних вимагають захисту даних паролів від несанкціонованої зміни або пошкодження. Тобто використання перевірок цілісності даних, безпечних і стійких до втручання методів зберігання, а також надійних методів резервного копіювання та відновлення.

- Доступність: Принцип доступності передбачає, що тільки авторизовані користувачі можуть отримати доступ до своїх облікових записів. Якщо говорити про зберігання паролів, то організації повинні підтримувати системи керування паролями, і робити їх ще більш надійними та стійкими, щоб запобігти втраті даних і забезпечити безперервність обслуговування навіть у випадку технічних збоїв або кібератак.

- І останній принцип – підзвітність, який вимагає, щоб організації не лише дотримувалися стандартів захисту даних, але й демонстрували, що вони їх виконують. Дотримання цього принципу може бути у вигляді ретельного документування всіх правил безпеки, включаючи збереження паролів, регулярні аудити безпеки та оцінки вразливостей.

Отже першим нормативним документом який регулює правильність та безпечність зберігання паролів є General Data Protection Regulation (GDPR) або ж Загальний регламент про захист даних Європейського Союзу. Цей документ не містить чітких вимог щодо використання конкретних криптографічних алгоритмів для зберігання паролів, але Принципи GDPR безпосередньо впливають на те, як паролі, що відносяться до категорії персональних даних, повинні оброблятися в межах ЄС і будь-якою організацією, яка обробляє дані резидентів ЄС, незалежно від місцезнаходження цієї організації.

Наприклад, у 5 статті GDPR вимагається, щоб персональні дані оброблялися "у спосіб, що забезпечує належну безпеку... з використанням відповідних технічних чи організаційних заходів" [1]. Це правило, якщо його застосовувати до зберігання паролів, вимагає впровадження надійних заходів

безпеки, які виходять за рамки звичайного дотримання на рівні організаційних заходів.

У 25 статті GDPR вимагається від організацій інтегрувати принципи захисту даних у структуру своїх систем і процесів [2]. Для зберігання паролів це означає організаційне та технічне зобов'язання розробляти системи, які за своєю суттю запобігають зберіганню паролів у відкритому вигляді і які використовують стійкі методи хешування.

Також у 32 статті є вимоги, що тримачі даних та ті, хто їх оброблюють, повинні впроваджувати «відповідні технічні та організаційні заходи» для забезпечення відповідного рівня безпеки, а саме використання псевдонімізації та шифрування як приклади цих дій [3].

Псевдонімізація зазвичай пов’язується із заміною прямих ідентифікаторів, а її основна ідея — перетворювати дані у незрозумілий для сторонніх вигляд без відповідного ключа розшифрування [4]. Схожий принцип застосовується до хешування паролів. Тому використання алгоритму хешування в поєднанні з унікальною сіллю, можна вважати формою псевдонімізації. Адже без конкретної солі або знань параметрів алгоритму зловмисник не зможе відновити оригінальний пароль, і викрадені хеші стають для нього фактично непотрібними. Стосовно впровадження та підтримки інфраструктури зберігання паролів, у 32 статті наголошується на постійній необхідності забезпечення конфіденційності, цілісності, доступності та стійкості систем обробки та зберігання.

Також варто згадати про інший міжнародний стандарт, дуже відомий та дуже використовуваний у більшості організацій – ISO/IEC 27001 (Системи управління інформаційною безпекою). Так, цей стандарт не є юридично обов’язковим, як наприклад закони чи нормативні акти, бо він має більш рекомендаційний характер але він надає комплексну основу для створення, впровадження, підтримки та постійного вдосконалення системи управління інформаційною безпекою.

ISO/IEC 27001 у своєму додатку A, має перелік контролів, які бажано щоб були імплементовані у системи організації для запобігання появи ризиків. Серед

цих контролів можна виділити контролі доступу, які надають рекомендації по управлінню паролями та їх зберіганню [5]. Серед цих рекомендацій для управління та зберігання паролів є запобігання витоку чутливих даних, тобто створення умов, щоб паролі не могли бути отримані зловмисниками. Також описані рекомендації стосовно використання стійких криптографічних методів, як для зберігання даних, так і для їх передачі.

Українське законодавство також має нормативну базу для регулювання процесів управління та зберігання паролів. Більшість з них ґрунтуються або відображають багато основних принципів, які містяться в GDPR.

Першим законом, який регулює питання управління та зберігання паролів є Закон України "Про захист персональних даних". Наприклад у 6 статті є вимога, щоб обробка персональних даних проводилася лише для чітко визначених, законних цілей, і зібрані дані повинні бути достатніми, доречними та не надмірними для цих цілей. Крім того, у статті вимагається, щоб особисті дані оброблялися захищеним чином [6]. Обидві ці вимоги є безпосередньо застосовувані до зберігання паролів, оскільки це забороняє збір непотрібних даних, пов’язаних з паролями, а також вимагає їхнього безпечного оброблення.

Також, у 24 статті цього ж закону є вимоги до тримачів та обробників даних, щоб вони вживали організаційні та технічні заходи, для запобіганню втраті, знищенню та несанкціонованому доступу до даних. Як і в GDPR, не вимагається конкретні процедури, лише використання стійких криптографічних рішень і належного захисту паролів.

Не менш важливими є закони "Про захист інформації в інформаційно-комунікаційних системах" та "Про основні засади забезпечення кібербезпеки України". У першому визначаються загальні засади інформаційної безпеки, включно з вимогами до автентифікації та контролю доступу [7]. Він є більш старим за сучасні підходи до зберігання паролів, але закладає базовий правовий фундамент у цьому питанні. Другий закон встановлює юридичні та організаційні стандарти кібербезпеки [8]. У ньому розглядаються вимоги щодо захисту

критичної інфраструктури від несанкціонованого доступу та безпечного зберігання даних.

Державна служба спеціального зв'язку та захисту інформації України (ДССЗЗІ) є основними регуляторними органом у сфері технічного та криптографічного захисту інформації. ДССЗЗІ має право видавати нормативні акти та стандарти, які можуть містити конкретні технічні вимоги до зберігання інформації, чутливих даних і звичайно паролів. Ці вимоги є особливо найбільш важливими для державних органів і об'єктів критичної інфраструктури.

Також варто уваги, окрім загальних законів про захист даних, згадати спеціальні регулювання в певних галузях. Одним з таких є Стандарт безпеки даних індустрії платіжних карток (PCI DSS) – набір стандартів безпеки, обов'язковий до виконання будь-якими організаціями, які обробляють, зберігають або передають дані власників карток або через ці організації відбуваються будь-які операції банківської системи. У PCI DSS є вимоги які стосується автентифікації, правильного використання паролів, а також захисту даних власників карток, і, хоча вони не вимагають використання конкретних алгоритмів захисту, проте вони вимагають впровадження стійких політик щодо безпечного зберігання облікових даних їх користувачів [9].

1.2 Типові загрози та вразливості при використанні паролів

Вразливості паролів – це слабкі місця, які можуть порушувати безпеку систем, маючи вплив як на окремих осіб, так і на організації. Ці слабкі сторони проявляються протягом усього життєвого циклу пароля, від його створення до зберігання та використання.

1.2.1 Основні вразливості паролів

Однією з найбільших проблем кібербезпеки залишаються слабкі паролі та їх повторне використання [10]. Загалом існують методи підвищення безпеки

паролів, але адміністратори систем та користувачі часто вирішують не використовувати їх.

Люди, зазвичай надають перевагу легкому запам'ятовуванню, а не безпеці, і саме тому часто вдаються до передбачуваних і вгадуваних паролів. Паролі з такої категорії є досить слабкими та їх легко або вгадати або зламати за допомогою відповідного програмного забезпечення. Для цього і використовують словники – файли з переліком найбільш розповсюджених паролів, що включають не лише загальноприйняті англійські слова, але й сленг, іншомовні слова і навіть назви брендів. Наприклад, словники можуть зберігати прості слова, такі як «security», «password» або «welcome», або ж трохи складніші фрази, як наприклад «Summer2024» і т.д.

Аналіз витоків різноманітних баз даних, а також словників паролів може надавати детальну інформацію про тенденції користувачів щодо вигадування або створення нового паролю. Наприклад, коли користувача просять створити пароль із малих і великих літер, користувачі, швидше за все, він буде використовувати лише першу літеру свого пароля у верхньому регістрі [11]. Коли система вимагає від користувачів додати в пароль цифри та/або спеціальні символи, вони, швидше за все, використовуватимуть цифрові послідовності, такі як «123», повторення чисел «111», числа, що мають певне значення, наприклад «314» (число π), або використовуватимуть заміну букв, наприклад символ «@» замість літери «a» та цифру «1» замість літери «i» [11].

Також вартим згадування є веб-сторінки, де є можливість перевіряти, скільки часу знадобиться, а й отже наскільки пароль є легким або складним, щоб зламати пароль за допомогою атаки грубої сили. Якщо, наприклад, ввести 8-значний пароль з цифрами, великими/малими літерами та спеціальними символами згенерований за допомогою іншого веб-сервісу, то на сайті *How Secure Is My Password?* комп'ютер зможе зламати такий пароль за приблизно 8 годин. Якщо ввести 16-значний пароль, який використовує лише малі літери та пароль є набором звичайних англійських слів, то результат вже буде 34 тисячі років [12].

Причина такої різниці – математика, бо для цього потрібно вирахувати ентропію. Але ентропія паролів вигаданих комп’ютером та людиною відрізняється. Зазвичай для пошуку ентропії для паролів, що були створені комп’ютером використовується загальна формула, де потрібно знайти квадратний логарифм від піднесення у степінь кількість можливих допустимих значень на довжину пароля (1.1).

$$E = log_2 (M^N) = N \times log_2 M \quad (1.1)$$

де E – значення ентропії;

M – кількість допустимих значень;

N – довжина паролю.

Для підрахунку ентропії паролів, що були вигадані людиною, використовується не загальна формула, для пошуку ентропії, а підхід, який був запропонований американським інститутом NIST [13]:

- ентропія першого символу пароля складає 4 біта;
- ентропія наступних семи символів пароля складає 2 біта на символ;
- ентропія символів з 9-го по 20-го складає 1,5 біта на символ;
- всі подальші символи мають ентропію 1 біт на символ;
- якщо пароль містить символи верхнього регістра і неалфавітні символи, то його ентропія збільшується на 6 біт.

Після того, як ентропія була знайдена, можна спробувати вирахувати час для зламу паролю, вигаданого людиною, для цього потрібно використовувати формулу 1.2 [14].

$$t = \frac{2^{E-1} \times T}{N} \quad (1.2)$$

де t – час, витрачений на злам паролю;

E – ентропія паролю;

T – час відповіді системи на запит (визначається в мілісекундах);

N – кількість одночасних спроб зламу паролю.

Отже, за допомогою цієї формули будь-хто може спробувати вирахувати час, який потрібен для того, щоб його пароль був зламаний.

У таблиці 1.1 наглядно відображається різниця у зламі певних паролів які складаються з: тільки з 5 малих літер, з 6 малих літер, малих літер та цифр, великих і малих літер.

Таблиця 1.1

Час для злому паролю, що складається з 5 малих літер, з 6 малих літер, малих літер та цифр, великих і малих літер.

5 малих літер	**Час для зламу, с**	**6 малих літер**	**Час для зламу, с**	**Малі літери та цифри**	**Час для зламу, с**	**Великі та малі літери**	**Час для зламу, с**
bales	57	bennis	1457	abas34	9944	Chough	73235
candy	58	daddle	1705	a346be	11524	Flabby	74221
delta	52	incite	1768	bute90	9567	Imagos	77564
egads	56	milady	1623	blips2	10044	Legmen	72345
feign	55	odours	1636	cupola	11116	Oafish	69783
peare	60	phenom	1554	5doggy	11167	Quiche	71864
taboo	67	quaked	1592	epm4t6	11591	Sebums	73455

Як було згадано вище, окрім слабкості паролів, існує інша їх вразливість – а саме використання одного й того самого пароля для створення облікових записів у різних веб-сторінках. Тобто у випадку, якщо хоча б одна з цих систем була зламана, то користувач, що використовував той самий пароль, і в інших застосунках, підлягає до компрометації. А це може відкрити доступ зловмисникам як до банківських облікові записів людини, так і, можливо, до робочих акаунтів і т.п.

Навіть якщо придумати найнадійніший пароль, він може стати небезпечним, якщо був використаний на кожному веб-сайті та в кожному додатку, якими користується людина. Враховуючи кількість глобальних витоків даних,

існує велика ймовірність того, що навіть цей "найнадійніший" пароль було скомпрометовано, завдяки неналежному дотриманню правил та вимог безпеки на деяких сайтах.

У 2023 році за статистикою BitWarden (менеджер паролів), серед їх користувачів близько 85% використовують один і той самий пароль для входу до різних облікових записів [15]. Або ж за опитуванням Google у 2019 році більш ніж 50% повторно використовують однакові паролі для входу до більшості веб-застосунків [16].

Існує ще одна дуже важлива вразливість при використанні паролів – те, як їх зберігають веб-застосунки після реєстрації на них користувачів. Більшість веб-додатки не зберігають незашифровані паролі. Тому, у випадку витоку даних, скомпрометованим буде лише хеш пароля. Однак у багатьох випадках хеші не є безпечними. Велика кількість ненадійних веб-додатків може використовувати старі алгоритми хешування, які легко скомпрометувати, наприклад, MD5 або SHA-1, для яких можна використовувати заздалегідь обчислені райдужні таблиці для швидкого перебору цих хешів.

Також небезпечним є відмова від використання солей або неправильне їх використання. При їх відсутності можна експлуатувати райдужні таблиці, або ж використання солі з ентропією менше 16 байтів також зменшує надійність та безпечність зберігання.

У всіх цих випадках, коли веб-застосунки відмовляються від використання надійних методів зберігання паролів, зловмиснику потрібно набагато менше часу, щоб підібрати пароль на основі хешу.

На зараз не існує способу перевірити повторне використання пароля у всіх веб-додатках, де користувач зареєстрований. Тому потрібно привчати себе дотримуватися простих вимог до створення надійного паролю: символів > 12, використовувати як великі так і малі літери, а також спеціальні символи і числа та найголовніша – не реєструватися на всіх неперевірених сайтах, але якщо дуже потрібно, то в такому випадку використовувати відмінний від основного пароль.

Загалом, у всьому вище згаданому, при будь-яких умовах, з'являється найголовніша вразливість для паролів – людський фактор. Чим простіший пароль вигадує людина, тим більша вірогідність того, що його легше зламати.

1.2.2 Загрози властиві при використанні паролів

Автентифікація за допомогою паролів є одним із найбільш поширених методів захисту інформації. Більшість систем контролю доступу все ще потребують парольного захисту, незважаючи на появу альтернативних технологій, таких як біометрична ідентифікація та багатофакторна автентифікація. Тому, як і для всього, що може існує в інтернеті або здатне надавати доступ до чутливих даних, існують загрози розкриття цієї інформації.

Нижче наведено основні з них:

- Phishing
- Атака грубої сили
- Man in the middle

А тепер більш детально про кожну з цих загроз.

Фішинг – це вид кібератак, що використовує електронні листи, текстові повідомлення, телефонні дзвінки або веб-сайти для шахрайства, щоб за допомогою обману отримати від людей їх конфіденційні дані, завантажити шкідливе програмне забезпечення на їх пристрої або іншим чином отримати з них вигоду [17].

Фішинг є одним з підвидів соціальної інженерії. Інші кібератаки, які зазвичай націлені на мережі та ресурси людей або організацій, сильно відрізняються від атак соціальної інженерії, оскільки вона використовує людські помилки, неіснуючі історії для заплутування людини або тактику тиску, щоб маніпулювати жертвами, змушуючи їх ненавмисно створити умови для завдання шкоди собі або організаціям, в яких вони працюють.

При звичайному використанні фішингу, зловмисник видає себе за людину, якій жертва може довіряти, наприклад, за колегу з місця роботи, керівника,

авторитетну особу або інших представників, які з першого погляду можуть здатися абсолютно нормальними та не викликати підозр. Вибравши за кого буде себе видавати, зловмисник надсилає жертві повідомлення, в якому просить оплатити рахунок, взяти для нього кредит, перейти за посиланням або виконати якусь іншу дію.

Довірившись тому, хто надіслав повідомлення, користувач виконує інструкції і потрапляє прямо в пастку від зловмисника. Надіслане ним посилання на “рахунок” може вести безпосередньо на веб-сторінку, за допомогою якої зловмисник може інсталювати на пристрій користувача програму-вимагач або інше шкідливе програмне забезпечення. Також перейшовши за посиланням користувач може потрапити на веб-сайт, за допомогою якого зловмисник може викрасти чутливі дані користувача: номери кредитних карток, банківських рахунків, облікові дані для входу в системи на яких зареєстрований користувач або інші персональні дані.

Фішинг можна вважати однієї з найбільш значимих загроз компрометації для всіх та для всього, що має підключення до інтернету, в тому числі фішинг є загрозою і для розкриття паролів. Це все стає можливим оскільки він використовує людський фактор, а не технічні вразливості систем. Завдяки чому зловмисникам не потрібно безпосередньо отримувати доступ до систем або вигадувати нові способи для обходу інструментів кібербезпеки. Все що потрібно зловмисникам, це обманом змусити людей, які мають доступ до того що їм потрібно – грошей, конфіденційної інформації чи чогось іншого і – виконати брудну роботу замість них.

Зловмисники, що займаються фішингом, можуть працювати як одинаки або цілими злочинними угрупованнями. Фішинг у них буде використовуватися для багатьох зловмисних цілей, як наприклад крадіжку особистих даних, отримання даних кредитних карток, крадіжку грошей, шпигунство або для збору компромату на людину з метою її залякування, погрожування тощо.

Фішинг – це загальне визначення кібератаки, тому, в залежності від того, який метод для реалізації вибирають зловмисники, і буде відрізнятися сам

фішинг. Кожен з цих підвидів буде відрізнятися тим, наскільки зловмисник може бути креативним та витонченим. Видів фішингу існує велика кількість, але основними можна вважати [18]:

Цілеспрямований фішинг або Spear phishing

Вид фішингу, який на відміну від загального, коли використовують тактику спаму на електронну пошту, щоб мати можливість атакувати тисячі людей одночасно, націлений на конкретних людей. При такому фішингу зловмисники додають до електронних листів імена, посади, робочі номери телефону та іншу інформацію, що переконає людину та змусить одержувача повірити, що відправник якимось чином знайомий з ним особисто або професійно. Spear phishing зазвичай використовується злочинними угрупованнями, які мають ресурси для дослідження та реалізації такої форми атаки.

Whaling

Whaling – вид цілеспрямованого фішингу, який націлений на керівників та інших високопосадовців («китів» звідки і пішла така назва). Такі особи, як правило, можуть мати доступ до конфіденційних корпоративних даних, тому співвідношення ризиків та винагороди є значно вищим. Фішинг спрямований на керівні посади знову таки використовується просунутими злочинними угрупованнями, які мають для цього ресурси, щоб здійснювати подібні атаки.

Вішинг або Vishing

Вид фішингу, відомий як голосовий фішинг. При вішингу зловмисник шляхом спеціальних маніпуляцій відображає реальний номер телефону певної компанії, до якої може бути певний рівень довіри, наприклад, банку або податкової служби. Після цього зловмисник починає відігравати роль керівника або іншої посадової особу і використовує методи та тактики соціальної інженерії, щоб жертва сплатила грошей, які нібито вона заборгувала цій організації. Вішинг також може у вигляді розсилки голосових повідомлень з проханням передзвонити на певний номер. Передзвонивши, жертву обманом змушують ввести власні особисті дані або реквізити її рахунку.

Смішинг або Smishing

Smishing – один з видів звичайного фішингу, який використовує фальшиві SMS-повідомлення, щоб досягнути поставленої цілі. У цьому випадку зловмисники зазвичай видають себе за провайдерів бездротового зв'язку і надсилають повідомлення з пропозицією «безкоштовного подарунка», для якого потрібно буде ввести власні банківські дані, або з проханням оновити інформацію про кредитну картку для продовження дії тарифу. Деякі зловмисники можуть відправляти SMS-повідомлення від імені поштових служб, з ціллю отримати гроші від жертви під виглядом оплати певної суми грошей для отримання замовленої посилки.

Наступна загроза – атака типу Man in the middle.

Man in the middle (MITM) – це вид атаки, у якій зловмисник втручається в обмін даними між користувачем і сервером. Він отримує змогу читати та змінювати інформацію, що надсилається або приймається [19]. Коли людина вводить пароль, цей пароль проходить через систему зловмисника. Унаслідок цього конфіденційні дані стають доступними сторонній особі. Ризик зростає, якщо з’єднання не використовує шифрування або має помилки в налаштуваннях безпеки.

Зловмисник може підробити точку доступу Wi-Fi, щоб перехопити трафік. У такому випадку користувач бачить знайому назву мережі й підключається до неї. Водночас зловмисник працює як псевдо-шлюз. Уся передана інформація, зокрема паролі, проходить через підроблену точку. Якщо на сайті немає правильного налаштування SSL або TLS, зловмисник читає й викрадає дані. Іноді зловмисник може підмінити сертифікати або використати DNS-підробку, щоб спрямувати жертву на фейковий ресурс.

Коли йдеться про паролі, MITM-атака загрожує тим, що людина не завжди помічає підміну. Адреса ресурсу може виглядати як оригінальна, а сторінка авторизації не викликає підозр. Зловмисник здатен змінювати поля введення. Він може додавати форми, що вимагають повторного підтвердження пароля або інших даних. Користувач довіряє цьому процесу й сам розкриває конфіденційну

інформацію. Таким чином паролі потрапляють у руки сторонньої особи, яка може використовувати їх для входу в облікові записи жертви.

MITM-атака вже більше залежить від того, чи належним чином налаштована мережева безпека. Навіть дотримуючись всіх порад, щодо запобіганню реалізації цієї вразливості (уникання сумнівних Wi-Fi мереж і не ігнорування попередження про загрозу безпеці), MITM-атака все ще залишається небезпечною. Оскільки зловмисник, якщо йому це дуже буде потрібно, зуміє викрасти паролі, знайшовши слабкі місця в мережевій інфраструктурі або використавши недоліки у методах шифрування трафіку та самих паролів.

І останньою, але не менш важливою, є загроза від атак грубої сили або Brute force.

Атака грубої сили, яка також відома як атака методом проб і помилок – атака, що використовує заздалегідь визначені значення для багаторазового підбору даних від облікового запису людини, зокрема часто використовується для підбору паролів. Для цього використовують спеціальне програмне забезпечення, яке буде виконувати перебір різних значень доки не буде знайдено правильний пароль. Зазвичай для цього можуть використовувати або словники, файли з найбільш розповсюдженими паролями або паролями, що були отримані у результаті витоків даних, або просто підбираючи правильні значення. Користувачі, які звичайні слова як паролі, легко піддаються такому виду атаки. Грубий перебір не спрацює, якщо адміністратор або система самостійно заблокує обліковий запис користувача, у якому було перевищено ліміт введення неправильного пароля.

Цілями атаки грубої сили, як і для минулих загроз – викрадення персональних даних, як наприклад ключові фрази, паролі та інша інформація, що може бути пов'язана з користувачем та яка використовуються ним для доступу до облікових записів банків або інших важливих веб-застосунків.

Загалом існує кілька різновидів атак грубої сили. Кожен з цих видів використовує власні прийоми, щоб мати можливість отримати конфіденційні дані

жертви, але всі вони мають спільну мету – отримання цих самих даних. Ви можете стати жертвою таких поширених методів грубої сили [20]:

1. Прості атаки грубої сили. Зловмисники намагається вгадати облікові дані без допомоги програм або додаткових засобів. Використовуючи такий вид атаки, вони просто перевіряють найпростіші паролі. Наприклад, якщо пароль – guest@123», є велика вірогідність, що зловмисник може його вгадати.

2. Словникова атака (Dictionary attack). Вид атаки, коли зловмисники заздалегідь вже знають, наприклад, ім'я користувача (username), та підбирають можливий пароль, саме для цього користувача за допомогою спеціальних файлів, де зібрана велика кількість вже відомих паролів. Найбільш відомий такий словник – rockyou, містить більше 14 мільйонів різних паролів. Саме тому через використання словників цю атаку і називають словниковою. Вона є найпоширенішою різновидністю атаки грубої сили. Дехто може вважати її не зовсім повноцінною атакою грубої сили, але вона точно використовується як один із ключових елементів при злому паролів.

3. Гібридна атака грубої сили. Такий різновид атаки грубої сили поєднує в собі елементи підбору за допомогою словників і звичайного підбору символів.

4. Атака зворотним перебором (Reverse Brute Force Attack). Ця атака відрізняється тим, що зловмисники спочатку мають пароль жертви, і в результаті чого, вони перебирають величезний набір імен користувачів, доки не знайдуть відповідний, для якого підходить даний пароль. Зазвичай зворотні атаки грубої сили використовуються не для того, щоб отримати доступ до певного користувача, а для того, щоб перевірити отримані паролі з витоку даних на багатьох користувачах.

5. Наповнення облікових даних (Credential Stuffing). Зловмисники можуть мати у своєму володінні велику базу даних імен користувачів та паролів. Ці дані, використовуються ними, щоб намагатися увійти до різних облікових записів у різних веб-застосунках. Наприклад, велика кількість людей можуть

використовувати однакові паролі на багатьох ресурсах, що і дає змогу виконувати такий тип атаки.

Також метод підбору, який використовується в атаці грубої сили, може застосовуватися для підбору або знаходження відповідного хешу паролів, що зберігаються в базах даних. Наприклад, використовуючи сучасну ігрову відеокарту, наприклад, Nvidia GeForce RTX 3090, з допомогою програмного забезпечення для злому паролів hashcat1, зловмисник може перебирати понад 65×10^9 варіантів паролів в секунду для відносно слабкої функції хешування, наприклад, MD5. У випадку використання Bcrypt з 32 ітераціями хешування це число падає до 100 000 варіантів в секунду. Хоча у другому випадку забезпечується більший опір зловмиснику, слабкий пароль захешований Bcrypt все одно буде знайдений швидше, ніж надійний, який хешувався за допомогою MD5.

1.3 Класифікація сучасних методів зберігання паролів. Їх переваги та недоліки

Сучасні системи баз даних стикаються з необхідністю безпечного зберігання паролів. Якщо неавторизований користувач або ж зловмисник отримує доступ до збережених облікових даних користувачів веб-застосунку, в тому числі і паролів, він отримує доступ до ресурсу відповідного користувача. Тому є важливим вибір методу зберігання паролів у базах даних.

1.3.1 Сучасні методи зберігання паролів у базах даних

Перший і найбільш наївний та незахищений спосіб зберігати паролі у базах даних – це зберігати їх у вигляді простого тексту. У такого методу є декілька «переваг», які насправді дуже складно назвати перевагами. По-перше, таке зберігання робить неймовірно легкою перевірку пароля користувача під час входу в систему. По-друге, у випадку якщо користувач забув свій пароль, то

можна зробити функцію відправки пароля йому на електронну пошту. Звісно обидві «переваги» є дуже сумнівними, бо вони максимально спрощують роботу будь-якого зловмисника, якому потрібно облікові дані користувачів, оскільки йому ніяк не треба зламувати або підбирати паролі. Тому такий метод є найжахливішим рішенням зберігання паролів у базах даних і його взагалі не рекомендується використовувати.

Хешування є основним методом зберігання паролів. Хеш-функція (ХФ) приймає вхідний рядок будь-якої довжини та створює вихідні дані фіксованої довжини, які називаються хешами. Гарна ХФ є односторонньою, що означає, що початкові вхідні дані не можуть бути реконструйовані з хешу будь-якими можливими способами [21]. Однією з найперших широко поширених функцій була MD5, на зараз її було замінено на більш нові та безпечніші методи, такі як SHA-256, або SHA-3 [22].

Коли користувачі реєструються, системи хешує їхні паролі та зберігає результат у базі даних. Під час входу система повторно хешує введений пароль і порівнює цей новий хеш із збереженим. Якщо вони збігаються, це підтверджує, що користувач ввів правильний пароль. Ця схема не дозволяє зловмисникам, які викрадають базу даних, одразу бачити паролі користувачів, як наприклад у випадку коли вони зберігаються у вигляді звичайного тексту.

Наступний метод – використання солі. Сіль – це випадковий рядок, який зберігається в базі даних разом із хешем пароля. Він не є секретним, але надає унікальність кожному паролю. Зловмисники часто створюють великі набори попередньо обчислених хешів, відомих як веселкові таблиці. Ці таблиці зіставляють загальні паролі з їх хеш-значеннями. Вони дозволяють зловмиснику швидк1.о перевірити багато припущень. Саме тому щоб запобігти таким атакам, бажаним буде додавання до пароля солі в процесі хешуванням [22].

Коли користувачі реєструються або змінюють свої паролі, система генерує унікальну сіль. Потім система додає цю сіль до пароля в процесі його хешуванням. Таким чином, два користувачі, які обрали один і той самий пароль, матимуть різні хеші в базі даних, оскільки їхні солі відрізняються. Такий підхід

робить райдужні таблиці менш ефективними, оскільки зловмисник повинен обчислювати окремі таблиці для кожного значення солі. Вартість та складність побудови райдужної таблиці для кожної можливої солі є досить високою, навіть для сучасного обладнання.

Зараз використання солі при зберіганні паролів є базовою вимогою до будь-якого веб-застосунку. Саме по собі соління не сповільнює спроби підібрати відповідний хеш пароля зловмисником, але воно має важливу роль в процесі зупинки можливої атаки з використанням заздалегідь обчисленого хешу (райдужних таблиць) [22].

Також сучасними методами є алгоритми розтягування ключа. Ця техніка спрямованою на підвищення безпеки та надійності ключів, отриманих із паролів. Цей підхід особливо важливий для зміцнення ключів перед загрозою атак методом грубої сили, коли зловмисники намагаються дешифрувати ключі

У своїй основі розтягнення ключа – це процес перетворення слабкого паролю або ключа в більш безпечний і складний. Це досягається за допомогою певних алгоритмів, які хешують початковий ключ або пароль в декілька ітерацій. Цей повторний процес хешування подовжує ключ і робить його більш складним і унікальним [23]. Таким чином, ключ, який було отримано, має значно більшу стійкість до атак методом грубої сили, що забезпечує вищий рівень безпеки для криптографічних завдань.

Формування криптографічного ключа часто починається з паролю, що визначається користувачем. Хоча це зручно, такі паролі можуть не володіти необхідною надійністю для протидії складним кібер-атакам. У цьому випадку розтягнення ключа виступає як захисний механізм. Використовуючи хеш-функції протягом багатьох ітерацій на початковому ключі або паролі, цей метод ефективно збільшує довжину та складність ключа. Цей процес завершується створенням більш унікального та міцного ключа, що значно підвищує його стійкість до несанкціонованих спроб дешифрування.

Центральне місце в техніці розтягнення ключа займають алгоритми, що полегшують перетворення базових ключів у зміцнені версії. Деякі з них наведені нижче:

- ХФ PBKDF2, також відома як функція генерації ключів на основі паролів, є алгоритмом, який дуже полюбився за свою роль у покращенні безпеки ключів. PBKDF2 підвищує обчислювальні зусилля, необхідні для успішного злому, додаючи псевдовипадкову функцію до значення солі вхідного паролю або ключа, шо і робить його більш безпечним [24].
- ХФ bcrypt є популярним алгоритмом для хешування паролів у багатьох веб-застосунках, оскільки він добре захищений від атак методом грубої сили. Його адаптивність є його перевагою, оскільки вона дозволяє змінювати складність хешування відповідно до збільшення обчислювальної потужності, зберігаючи при цьому ефективність [25].
- Scrypt встановлює високий стандарт стійкості до паралельних обчислень та атак на основі графічних процесорів, оскільки він розроблений з метою бути вимогливим до пам'яті та процесора, що робить його доволі повільним порівнюючи з іншими [26].
- Argon2 є функцією хешування паролів, яка перемогла у конкурсі Password Hashing Competition (PHC) [27]. Argon2 – це функція, яка узагальнює сучасні досягнення у розробці безпекових рішень, що є дуже вимогливими до пам'яті, і може бути використана для хешування паролів та зберігання облікових даних та іншого [28].

Ще один метод зберігання паролів у базах даних – використання шифрування. Шифрування перетворює дані у формат даних, що досить складно причитати за допомогою ключа. Відновлення цих даних можливе лише за допомогою відповідного правильного ключа розшифрування [24]. Для зберігання паролів шифрування є менш поширеним, ніж хешування, оскільки воно створює можливість зворотного процесу, що суперечить меті одностороннього захисту паролів.

В залежності від вибору методу шифрування воно може збільшити накладні витрати при шифруванні або збільшити навантаження на систему, але не зважаючи на це все, шифрування більш надійно захищає будь-які дані в безпечному стані, особливо якщо зловмисник не має можливості отримати доступ до ключа розшифрування.

1.3.2 Переваги та недоліки обраних методів зберігання паролів

Будь-який метод чи алгоритм, яким би надійним та захищеним він не був, завжди буде мати свої переваги та недоліки. Нижче, я спробую виділити для кожного методу свої переваги та недоліки, які для них характерні.

Хешування

Переваги:

- Хешування є односторонньою функцією, що унеможливлює отримання вхідних даних за допомогою оборотного процесу.
- Велика підтримка більшістю мов програмування та фреймворків, які мають відповідні бібліотеки для стандартних хешів.
- Хешування є дуже швидким та легким алгоритмом, що зменшує навантаження на системи, якщо використовувати їх для зберігання паролів.

Недоліки:

- Існує велика імовірність, що зловмисники можуть вгадати пароль, хешуючи вхідні дані та порівнюючи їх зі збереженим хешем, або вже використовуючи попередньо підготовлені райдужні таблиці. Чим слабше та коротше був вибраний пароль користувачем, тим швидше відбувається підбір.
- Використання сучасного обладнання полегшує злам хешів, оскільки воно може обчислювати велику кількість хешів за секунду.

Використання солі

Переваги:

- Створює умови, що запобігають атаки за допомогою райдужних таблиць.

- Є легким методом для реалізації у більшості систем автентифікації, оскільки солі можна зберігати в окремому стовпчику або полі даних, що не вимагає значних змін в них.
- Унеможливлює ситуації, коли два користувачі з однаковим паролем будуть мати однаковий хеш в базі даних.

Недоліки:

- Наявність солі, не відміняє можливість застосування зловмисниками атаки грубої сили.
- При використанні солі, може виникати складність у її зберіганні, бо кожен користувач повинен мати власну унікальну сіль, а система, що відповідає за автентифікацію, повинна правильно обробляти цю сіль, а також сіль повинна бути завжди доступна для перевірки.

Алгоритми розтягування ключів

Переваги:

- Уповільнюється злам паролів зловмисниками, оскільки ці алгоритми вимагають повторення або виконання складних операцій, що зменшує кількість підборів хешів в секунду.
- Є влаштоване налаштування кількості ітерацій, що допомагає відповідати бажаному рівню безпеки.
- Більшість таких алгоритмів мають автоматичне створення та використання солі

Недоліки:

- Ці алгоритми є більш вибагливими до ресурсів системи, що може спричинити більше навантаження і в результаті чого довший час очікування для автентифікації користувача.

Шифрування

Переваги:

- Без знання ключа розшифрування, зловмиснику буде майже неможливо отримати розшифровані дані.

- Використання математичних принципах робить шифрування складнішим до зламу.

Недоліки:

- У випадку компрометації ключа шифрування, всі зашифровані ним паролі стають вразливими.
- Створюються труднощі в розробці системи, бо виникає необхідність попередньо створювати умови для зберігання та обробки ключі.

1.4 Висновки до першого розділу

У цьому розділі були розглянуті нормативні вимоги та стандарти безпеки, що регламентують зберігання паролів у базах даних веб-застосунків. Такі нормативні або національні законодавчі акти, як GDPR, ISO/IEC 27001 зобов'язують тримачів систем гарантувати конфіденційність, цілісність і доступність даних користувачів.

Також був проведений аналіз та визначено головні загрози і вразливості, при використанні та збереженні паролів. Найбільш важливими та небезпечними можна вважати фішингові атаки, методи грубої сили, зловживання повторним використанням паролів і відсутність належних захисних механізмів. незалежно від системи захисту – людський фактор залишається одним із найбільших ризиків.

Сучасні підходи забезпечення цілісності збережених паролів у базах даних головною метою мають ускладнення роботи зловмисникам. Хешування з унікальною сіллю, ускладнює процес пошуку правильного значення з попередньо обчисленими хешами. Алгоритми розтягування ключів уповільнюють перебір можливих значень. Шифрування надає можливості використання додаткового рівня захисту, але разом з цим потребує коректного управління ключами, щоби уникнути масового розкриття всієї інформації у випадку їх розсекречення.

РОЗДІЛ 2
АНАЛІЗ АЛГОРИТМІВ ЗАХИСТУ ДЛЯ ЗБЕРІГАННЯ ПАРОЛІВ

2.1 Ключові принципи у захисті паролів

Зазвичай паролі є першим та основним бар'єром між зловмисником та конфіденційними даними користувача. Тому розробники систем приділяють особливу увагу способам безпечного зберігання та обробки паролів. Криптографічні підходи допомагають знизити ймовірність компрометації та спростити організацію безпечних процедур аутентифікації.

2.1.1 Загальні технічні особливості збереження паролів

Однонаправленість означає, що функція перетворює вхідні дані будь-якого розміру на вихідні дані певного статичного розміру (наприклад 256 біт, 512 біт, т.п.) таким чином, що відновлення початкового значення є надзвичайно складним. Цю властивість забезпечують криптографічні хеш-функції. Коли сервер зберігає паролі у хешованому вигляді, він насправді унеможливлює пряме відтворення паролів з бази даних. Адміністратори не бачать текст паролів. Зловмисник, який викрав базу, також не отримає паролі безпосередньо. Йому доведеться виконувати перебір чи аналіз відбитків, що значно ускладнює атаку.

У той самий час симетричне або асиметричне шифрування вимагає наявності ключа для розшифрування. Зберігаючи ключ в системі або взагалі разом з паролем, з'являється можливість, що зловмисник може його викрасти. В такому випадку він отримує доступ до всіх збережених паролів у відкритому вигляді.

Використовуючи хеш такої ситуації не виникне, адже відновлення паролю з самого хешу є неможливим. Хешування закриває можливість небажаної

цікавості перегляду паролів з боку службового персоналу й захищає від загроз внутрішніх зловмисників.

Але при використанні звичайних криптографічних хешів (SHA-256, SHA-512), які працюють досить швидко, виникає вразливість. Зловмисники використовують цю саму швидкість для підбору дуже великої кількості варіантів паролів, використовуючи для цього паралельний перебір можливих значень. Тому замість звичайних хеш-функцій бажано використовувати більш безпечні алгоритми: bcrypt, Argon2 і т.д. Вони збільшують вартість паралельного перебору за часом та ресурсами, необхідними для цього, вони є досить адаптивними, тобто чим краще потужність апаратного забезпечення – тим краще стає захист за рахунок збільшення ітерацій або параметрів пам’яті.

Але такі алгоритми також мають свої недоліки. У більшості випадків вони хешують дані набагато довше, аніж звичайні хеш-функції, що, наприклад, при великому обсязі одночасних спроб автентифікації може викликати великі затримки для користувачів. Також вони є більш вибагливими до обчислювальних ресурсів, що також інколи може бути головною причиною відмовитися від них (у випадках коли я жорсткі обсяги доступних обчислювальних ресурсів).

Не менш важливим принципом при виборі методів для захисту паролів є сіль, тобто випадкові дані, які система додає до пароля перед хешуванням. Це робить кожен пароль унікальним, навіть якщо користувачами був обраний ідентичний пароль або парольна фраза. Для цього бажано щоб сіль була достатньо довгою, щоб зробити майже неможливою ситуацію коли її значення повторюється.

Як було вже сказано раніше в роботі, найголовнішою причиною, навіщо використовувати сіль – запобігання використання зловмисниками райдужних таблиць, де заздалегідь обчислені хеші для поширених паролів

Також вибираючи, методи для збереження паролів, слід враховувати те, що зловмисники все ще можуть використовувати атаку грубої сили, тому бажано робити так, щоб якомога більше уповільнювати цей процес для зловмисників. Для цього підійдуть всі методи, що були описані раніше у цій роботі.

Використовуючи їх один з одним або поодинці збільшує витрати часу на кожний пароль суттєво, тим самим зменшуючи мотивацію зловмисника, навіть якщо буде використовуватися підбір паролів за словником з найбільш популярними паролями.

2.1.2 Важливість окремого зберігання ключів при шифрування

Якщо все ж таки є необхідність використовувати шифрування, то потрібно дуже сильно попіклуватися про те, як і де зберігаються ключі розшифрування.

Симетричне шифрування, як зрозуміло з його назви, використовує один ключ для обох операцій: шифрування та розшифрування. Прикладом слугує AES. Якщо використовувати AES для зберігання критичних даних, то потрібно створити умови, при яких ключ буде захищений від несанкціонованого доступу, оскільки викрадення цього ключа повністю нівелює використання алгоритму шифрування, адже тоді зловмисник отримує можливість розшифрувати будь-яку зашифровану інформацію. Тому ключ симетричного шифрування повинен зберігатися у захищеному сховищі, до якого зловмисник не буде можливості потрапити [29]. Якщо ключ зберігається у текстовому файлі на сервері, то це створює значні ризики, бо існує вірогідність, що зловмисник може знайти цей файл, тим самим отримавши доступ до всіх зашифрованих даних.

Асиметричне шифрування, на відміну від симетричного, працює з двома ключами: відкритим (для шифрування даних) та закритим (для розшифрування) [30]. Відкритий ключ можна поширювати вільно, а закритий необхідно зберігати у захищених сховищах або інших безпечних місцях, куди зловмисник не зможе отримати доступ, бо отримавши доступ він отримає можливість розшифрувати всі дані, як і у випадку з симетричним шифруванням.

Окреме зберігання ключів шифрування поза базами даних збільшує захист системи, бо зловмиснику потрібно докладати подвійних зусиль. У такому випадку йому щоб розшифрувати дані потрібно спочатку отримати доступ до захищеної бази, де містяться зашифровані паролі, а також отримати доступ до

захищеного сховища, де зберігається ключ. У випадку коли перша частина виконалася, а друга залишаться без можливості отримання доступу до неї, то зловмисник не зможе отримати вхідні дані, тобто паролі.

2.1.3 Роль векторів ініціалізації при симетричному шифруванні

Вектор ініціалізації (IV) потрібен алгоритмам симетричного шифрування, які працюють у режимах шифрування блоків. IV задає початковий стан шифрувального процесу, щоб два однакові блоки даних за того самого ключа не перетворювалися на ідентичний шифротекст [31]. Це позбавляє зловмисника змоги відстежувати повторювані шаблони в зашифрованих даних.

Для збереження конфіденційності IV має бути випадковим або принаймні неповторним. Якщо його повторити, можливі атаки, що дають змогу зіставити окремі сегменти інформації або відтворити структуру повідомлень.

IV не є таємним ключем. Його можна передавати разом із зашифрованим текстом, бо він лише впливає на початковий стан шифру.

Під час реалізації симетричного шифрування критичних даних (не тільки паролів) рекомендуються генерувати IV на кожну операцію шифрування. Система, що за це відповідає може зберігати IV у полі, яке йде разом із зашифрованими даними. Зберігання IV окремо від ключа не є обов’язковою вимогою, але рекомендуються слідкувати за тим, щоб зловмисник не мав можливості навмисно підмінити IV задля криптоатаки.

За відсутності IV або при використанні сталого IV ті самі блоки «відкритих» даних будуть щоразу шифруватися однаково, розкриваючи структуру повідомлень. Якщо це стосується паролів або сесійних ключів, повторення робить систему вразливою. IV усуває ці вразливості, бо кожен блок отримує унікальний початковий стан [32]. У підсумку зловмисник не може напряму відшукати однакові фрагменти даних.

2.2 Огляд алгоритмів AES-256, SHA-512 та bcrypt

Перш ніж проводити порівняння між алгоритмами, що є складовими створеного нового методу зберігання паролів, з вже існуючими, бажано спочатку розібратися в тому яким саме чином вони працюють та виконують свої функції хешування або шифрування.

2.2.1 Алгоритм SHA-512

Перед тим як перейти до більш детального розбору саме SHA-512, спочатку потрібно декілька слів сказати взагалі про те, що таке хеш-функції та алгоритми хешування загалом (рис. 2.1.).

Хеш-функція – це математична функція, яка приймає рядок зі змінною кількістю символів, який також називають повідомленням, і перетворює його в рядок з фіксованою кількістю символів, або інша назва – хеш (хеш-значення) [33].

Алгоритм хешування, у свою чергу, це саме процес використання відповідної хеш-функції для перетворення вхідного повідомлення на хеш – рядок символів, що людина не зможе прочитати.

Рисунок 2.1 – Зображення загального виду алгоритмів хешування.

SHA-512, або Secure Hash Algorithm 512 – це алгоритм хешування, який використовується для перетворення тексту будь-якої довжини в рядок фіксованого розміру довжиною 512 біт (64 байти) [34].

SHA-512 – є одним з декількох алгоритмів в сімействі алгоритмів безпечного хешування (SHA). У 2001 році SHA-512 був опублікований Національним інститутом стандартів і технологій (NIST) як Федеральний стандарт обробки інформації США (FIPS) [35].

Цей алгоритм зазвичай використовується для хешування адрес електронної пошти, хешування паролів і перевірки цифрових підписів. SHA-512 також може використовуватися в технології блокчейн.

Для ХФ SHA-512 характерні наступні властивості [35]:

- Генерація хеш розміром 512 біт забезпечує дуже велику кількість можливих вихідних значень, що запобігає виникнення колізій.
- Непередбачуваність і незворотність хешу гарантується за допомогою модульного додавання та побітового обертання.
- Алгоритм обробляє вхідні дані довжиною до 2^{128} біт, що дозволяє працювати з різними обсягами інформації.
- SHA-512 використовує послідовний процес з кількома раундами, кожен з яких застосовує конкретні математичні функції. Використання такої структури посилює лавиновий ефект, що робить складнішим обчислення хешу навіть при зміні одного біту вхідних даних.
- SHA-512 є достатньо універсальним алгоритмом, який здебільшого використовується у криптографії, але його також можна застосовувати для перевірки контрольних сум або забезпечення цілісності даних.

А тепер перейдемо від більш загального до більш конкретного, а саме до того, що відбувається всередині хеш-функції алгоритму SHA-512.

Хеш-функція алгоритму SHA-512 складається з 4 основних етапів [36]:

- Форматування та підготовка вхідних даних.
- Ініціалізація хеш-буфера.
- Обробка повідомлення (саме процес хешування).
- Вивід хеш-рядка.

Етап форматування та підготовки вхідних даних

SHA-512 приймає на вході повідомлення довільного розміру, але не більше ніж 2^{128} біт (рис. 2.2). Це обмеження випливає з самої структури алгоритму. Відформатоване повідомлення складається з трьох частин: оригінального повідомлення, бітів заповнення та розміру оригінального повідомлення. Разом вони мають утворювати кратний 1024 біти блок, оскільки обробка відбувається блоками по 1024 біти.

Якщо у вхідного повідомлення не вистачає бітів, для того щоб утворити блок 1024 біт, то у такому випадку до повідомлення додаються біти заповнення для досягнення потрібної довжини. Для заповнення використовуються нульові біти з однією провідною одиницею (100000…000). Заповнення виконується навіть за наявності лише одного біту, який буде дорівнювати одиниці (рис. 2.3). Загальна довжина повідомлення повинна бути на 128 біт меншою за кратне 1024, щоб після додавання розміру повідомлення отримати розмір, кратний 1024 біти.

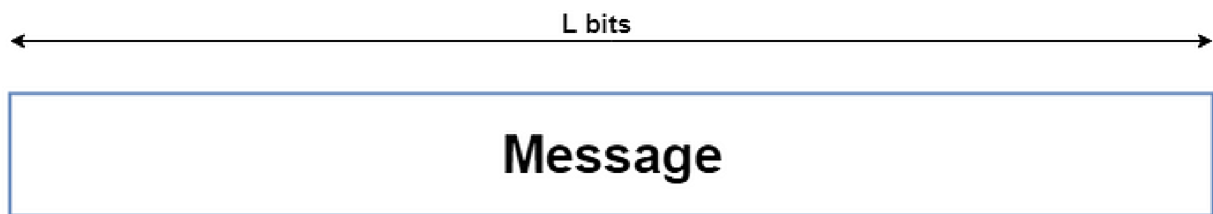


Рисунок 2.2 – Вхідне повідомлення довільно довжини.

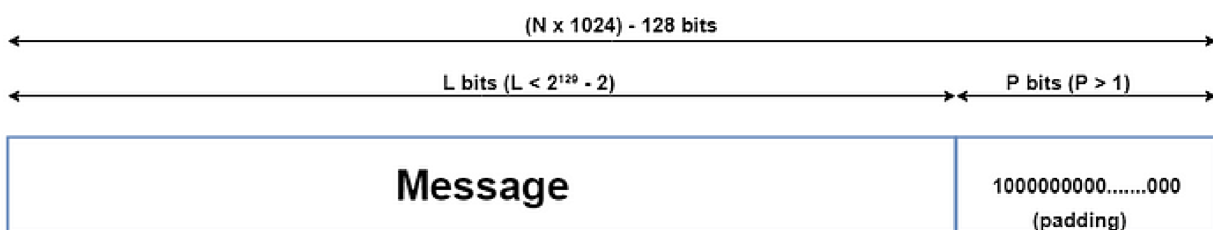


Рисунок 2.3 – Додавання бітів доповнення до повідомлення.

Після додавання бітів доповнення до повідомлення додається інформацію про його розмір, яка кодується у 128 біт (рис. 2.4) [36]. Саме це і створює обмеження на максимальний розмір вхідних даних: максимальне число, яке можна представити у 128 бітах, дорівнює $(2^{128} - 1)$ біт. З урахуванням хоча б одного біту заповнення, максимальний розмір оригінального повідомлення

становить $(2^{128} - 2)$ біти. Це обмеження практично не має значення, оскільки максимальне значення вхідних даних є надзвичайно великим ($2^{128} - 2$ = 340 282 366 920 938 463 463 374 607 431 768 211 454 біт = 4.2535333651173079517 Петабайт).

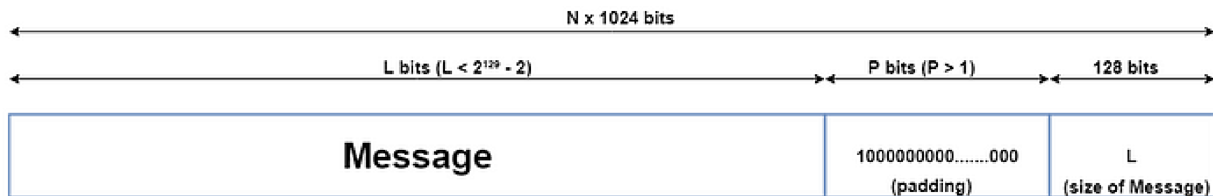


Рисунок 2.4 – Додавання довжини повідомлення.

Після того як біти заповнення та розмір повідомлення додані, вхідні дані повністю відформатовано та вони готові до подальших дій.

Етап ініціалізації хеш-буферу

Наступним після форматування вхідного повідомлення йде створення буферу ініціалізації. Алгоритм працює так, що він обробляє кожен 1024-бітний блок повідомлення, використовуючи результат попереднього блоку. Це створює проблему для найпершого 1024-бітного блоку, який не може використати результат попередньої обробки. Ця проблема вирішується використанням значення за замовчуванням для першого блоку – векторів ініціалізації, щоб почати процес хешування.

Враховуючи той факт, що кожен проміжний результат потрібен для оброблення наступного блоку, його треба десь зберігати для подальшого використання. Цим займається хеш-буфер. Він також тримає кінцевий хеш-значення всього процесу SHA-512 як останній з проміжних результатів.

Значення за замовчуванням (IV) яке використовується для старту обробки кожного 1024-бітного блоку, також записується до хеш-буфера на початку процесу обробки (рис. 2.5). Всього цих значень векторів ініціалізації 8 і беруться вони з перших 64 біт дробових часток квадратних коренів із перших восьми простих чисел (2, 3, 5, 7, 11, 13, 17, 19) [36].

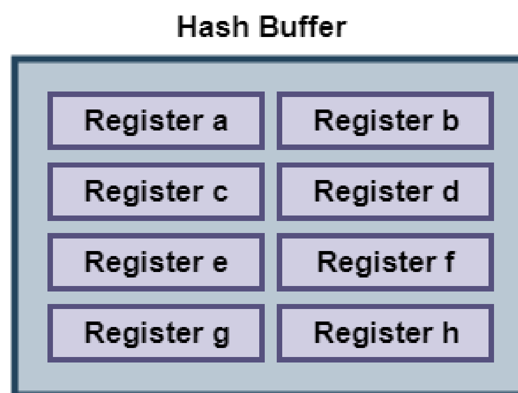
Чому саме 8? Тому що SHA-512 працює з 64-бітними словами. Оскільки хеш має довжину 512 біт, а 512 / 64 = 8, потрібно 8 таких 64-бітних слів (векторів

ініціалізації), щоб представити весь хеш. Кожен з восьми IV зберігає одну 64-бітну частину проміжного (або кінцевого) стану хешу.

Значення IV зберігаються у форматі big-endian, у якому старшим байтом слова є байт в молодшій (лівій) позиції.

До кожного блоку на початку заходять наступні значення:

- Вихід попереднього раунду (рядок довжиною 512 біт) або початкові значення, якщо це перший блок. Початкові значення це і є ті самі 8 векторів.
- Поточний блок повідомлення довжиною 1024 біт.



Initialization Vector

a = 0x6A09E667F3BCC908 b = 0xBB67AE8584CAA73B

c = 0x3C6EF372FE94F82B d = 0xA54FF53A5F1D36F1

e = 0x510E527FADE682D1 f = 0x9B05688C2B3E6C1F

g = 0x1F83D9ABFB41BD6B h = 0x5BE0CD19137E2179

Рисунок 2.5 – Хеш-буфер та вектори ініціалізації.

Етап обробки повідомлення або сам процес хешування

На цьому етапі найголовнішою є функція стиснення яка приймає як вхідні значення 1024 біт повідомлення та 512 біт попередньо обробленого повідомлення (або якщо це найперший блок, то початкові вектори ініціалізації).

Вхідний 1024-бітний блок повідомлення розбивається на 80 слів ($W_0 - W_{79}$) по 64 біти кожне. З яких перші 16 слів ($W_0 - W_{15}$) – це безпосередньо саме

повідомлення, розбите на 64-бітні частини. Решта 64 слова ($W_{16} - W_{79}$) обчислюються за рекурентною формулою, яка використовує попередні слова з додаванням, побітовими операціями (XOR, AND, NOT) і циклічними зсувами [37].

Основою функції стиснення є раунди. Всього їх 80. Кожен раунд на початку приймає до себе три речі: одне слово (W_i), результат попереднього раунду та константу SHA-512 (рис. 2.6).

Використання констант для кожного раунду є обов’язковим, оскільки вони забезпечують "випадковий" набір 64-бітових шаблонів, що має усунути будь-які закономірності у вхідних даних.

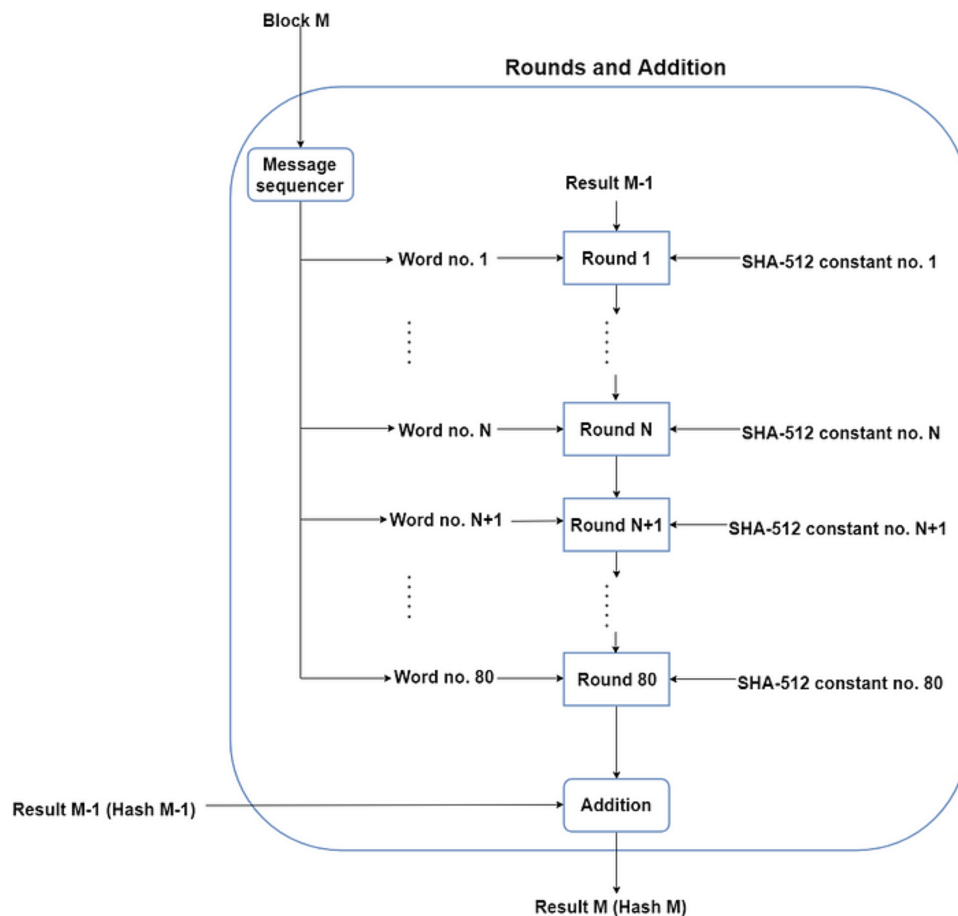


Рисунок 2.6 – Зображення загального виду процесу стиснення SHA-512.

Константи собою являють перші шістдесят чотири біти дробових частин кубічних коренів перших вісімдесяти простих чисел (2, 3, 5, 7, 11, 13, 17, 19 і так далі) [38]. А потрібно саме 80 констант тому що у процесі стиснення використовується 80 раундів. Константи зображені на рисунку 2.7.

```
k[0..79] := [ 0x428a2f98d728ae22, 0x7137449123ef65cd, 0xb5c0fbcfec4d3b2f, 0xe9b5dba58189dbbc, 0x3956c25bf348b538,
              0x59f111f1b605d019, 0x923f82a4af194f9b, 0xab1c5ed5da6d8118, 0xd807aa98a3030242, 0x12835b0145706fbe,
              0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2, 0x72be5d74f27b896f, 0x80deb1fe3b1696b1, 0x9bdc06a725c71235,
              0xc19bf174cf692694, 0xe49b69c19ef14ad2, 0xefbe4786384f25e3, 0x0fc19dc68b8cd5b5, 0x240ca1cc77ac9c65,
              0x2de92c6f592b0275, 0x4a7484aa6ea6e483, 0x5cb0a9dcbd41fbd4, 0x76f988da831153b5, 0x983e5152ee66dfab,
              0xa831c66d2db43210, 0xb00327c898fb213f, 0xbf597fc7beef0ee4, 0xc6e00bf33da88fc2, 0xd5a79147930aa725,
              0x06ca6351e003826f, 0x142929670a0e6e70, 0x27b70a8546d22ffc, 0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed,
              0x53380d139d95b3df, 0x650a73548baf63de, 0x766a0abb3c77b2a8, 0x81c2c92e47edaee6, 0x92722c851482353b,
              0xa2bfe8a14cf10364, 0xa81a664bbc423001, 0xc24b8b70d0f89791, 0xc76c51a30654be30, 0xd192e819d6ef5218,
              0xd69906245565a910, 0xf40e35855771202a, 0x106aa07032bbd1b8, 0x19a4c116b8d2d0c8, 0x1e376c085141ab53,
              0x2748774cdf8eeb99, 0x34b0bcb5e19b48a8, 0x391c0cb3c5c95a63, 0x4ed8aa4ae3418acb, 0x5b9cca4f7763e373,
              0x682e6ff3d6b2b8a3, 0x748f82ee5defb2fc, 0x78a5636f43172f60, 0x84c87814a1f0ab72, 0x8cc702081a6439ec,
              0x90befffa23631e28, 0xa4506cebde82bde9, 0xbef9a3f7b2c67915, 0xc67178f2e372532b, 0xca273eceea26619c,
              0xd186b8c721c0c207, 0xeada7dd6cde0eb1e, 0xf57d4f7fee6ed178, 0x06f067aa72176fba, 0x0a637dc5a2c898a6,
              0x113f9804bef90dae, 0x1b710b35131c471b, 0x28db77f523047d84, 0x32caab7b40c72493, 0x3c9ebe0a15c9bebc,
              0x431d67c49c100d4c, 0x4cc5d4becb3e42b6, 0x597f299cfc657e2a, 0x5fcb6fab3ad6faec, 0x6c44198c4a475817]
```

Рисунок 2.7 – Константи для раундів SHA-512.

Кожен раунд складається з певного набору математичних функцій, які є основними для хешування за допомогою SHA-512. Перелік цих функцій наведено у формулі 2.1 [36].

$$
\begin{gathered}
T_1 = h + Ch(e, f, g) + \left(\sum_{1}^{512} e\right) + W_t + K_t \\
T_2 = \left(\sum_{0}^{512} a\right) + Maj(a, b, c) \\
\sum_{0}^{512} a = ROTR^{28}(a) \oplus ROTR^{34}(a) \oplus ROTR^{39}(a) \\
\sum_{1}^{512} e = ROTR^{14}(e) \oplus ROTR^{18}(e) \oplus ROTR^{41}(e) \\
a = T_1 + T_2 \\
b = a \\
c = b \\
d = c \\
e = d + T_1 \\
f = e \\
g = f \\
h = g
\end{gathered}
\tag{2.1}
$$

де a, b, c, d, e, f, g, h – значення регістрів хеш-буферу;

t – номер раунду;

Ch (e, f, g) = (e AND f) (NOT e AND g) функція умови: Якщо e тоді f інакше g;

Maj (a, b, c) = (a AND b) (a AND c) (b AND c) функція є істинною лише тоді, коли більшість аргументів є істинними;

функція $ROTR^n(x)$ – круговий зсув вправо 64-бітового аргументу x на n біт;

W_t – 64-бітне слово, отримане з поточного 512-бітового вхідного блоку;

K_t – 64-бітна константа;

+ – додавання за модулем 264.

Також відображення процесу для кожного раунду алгоритму SHA-512 зображено на рисунку 2.8.

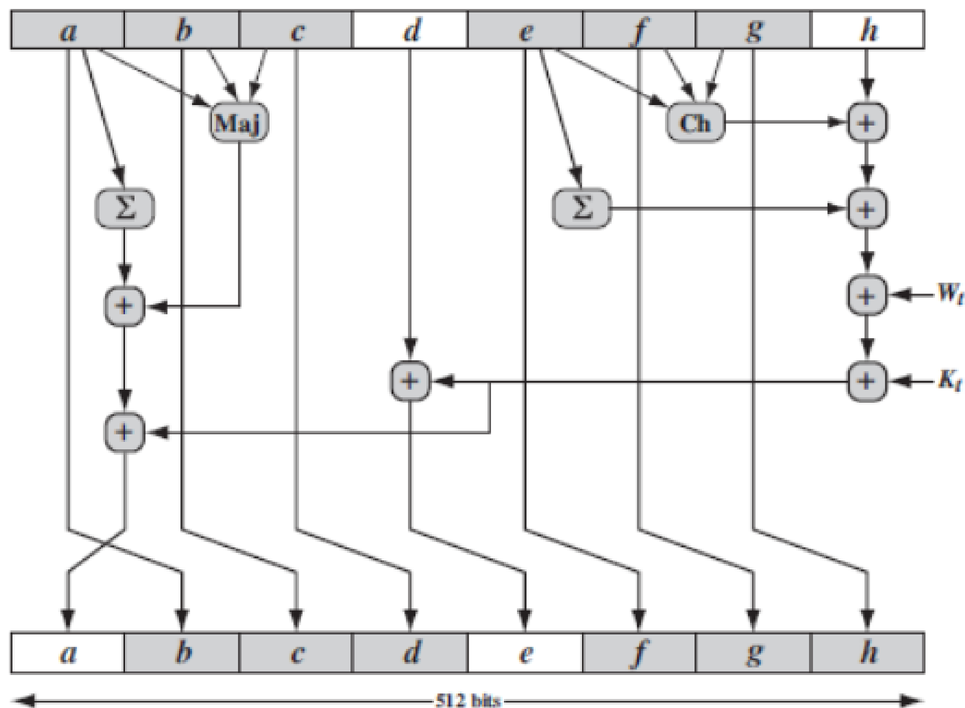


Рисунок 2.8 – Перелік дій в кожному раунді алгоритму SHA-512.

Етап виводу хеш-значення

Після оброблення кожного 1024-бітного блоку його проміжні результати стають вхідними даними для наступного. Коли завершується обробка останнього

1024-бітного блоку, отримується остаточне 512-бітне хеш-значення вихідного повідомлення. Загальний вид алгоритму SHA-512 можна побачити на рисунку 2.9.

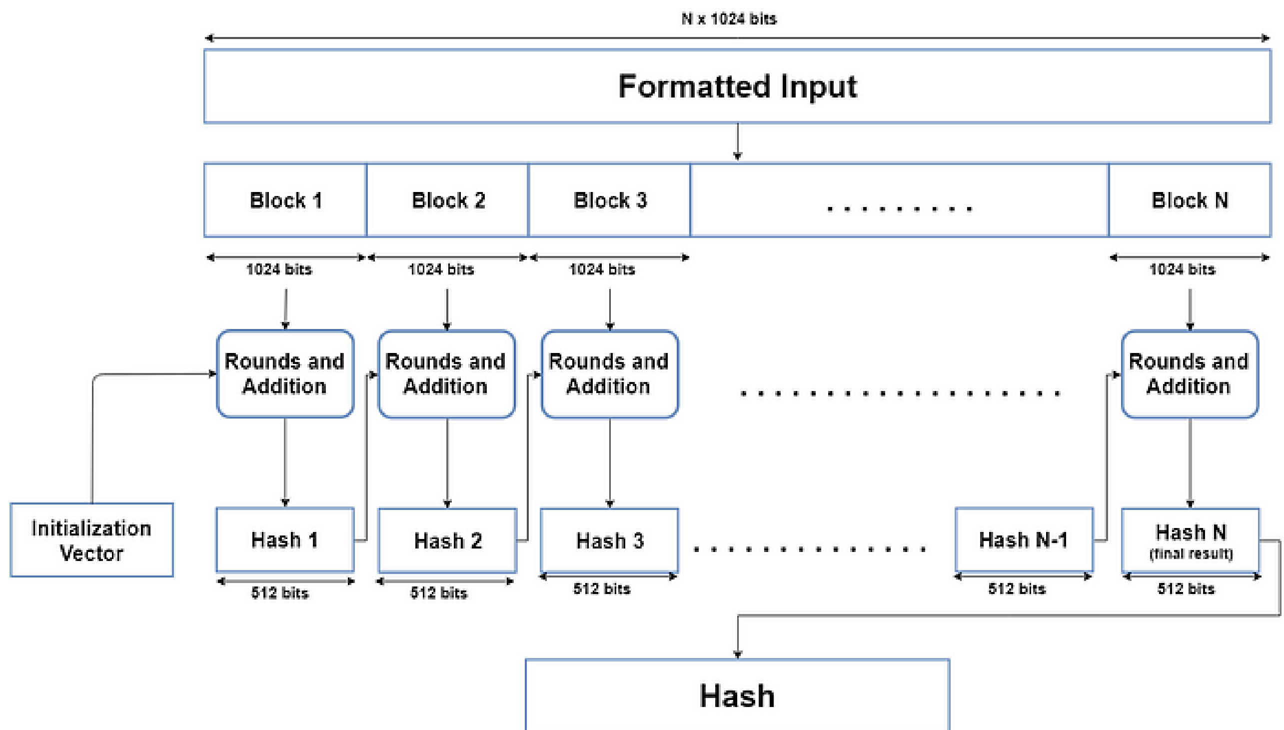


Рисунок 2.9 – Алгоритм хешування SHA-512.

2.2.2 Алгоритм шифрування AES

Advanced Encryption Standard (AES) – алгоритм шифрування, що став одним із найпоширеніших сучасних методів шифрування та дешифрування. Він є модернізованою версією алгоритму Rijndael – блочного шифра, розробленого Вінсентом Рейменом та Жоан Даеманом. Шифрування та дешифрування даних у AES відбувається за допомогою одного ключа, що і робить його симетричним алгоритмом [39]. Загальний вигляд алгоритму шифрування AES зображено на рисунку 2.10.

AES розроблено так, щоб забезпечити однакову ефективність як у апаратному, так і в програмному забезпеченні. Це досягається завдячуючи принципу заміни-перестановки, який використовується в проектуванні блочного шифра.

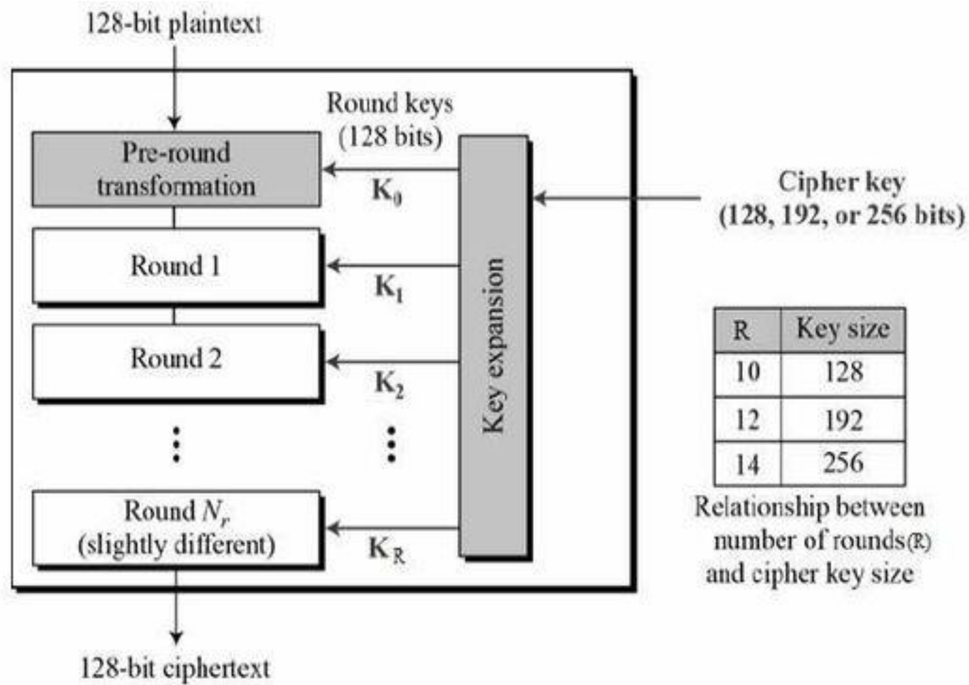


Рисунок 2.10 – Загальний вигляд алгоритму AES.

Оскільки блочний розмір у 64 біти, притаманний DES – алгоритму, що був стандартом до появи AES, виявився ненадійним через парадокс днів народження, у AES використовується блочний розмір у 128 біт.

За парадоксом днів народження (birthday paradox), коли кількість блоків досягає квадратного кореня від максимально можливої кількості, існує приблизно 50% шанс, що два чи більше блоків будуть однакові. Це значно спрощує дешифрування та може призвести до витоку інформації про вміст повідомлення. Таким чином, зашифрованими за стандартом DES можна безпечно передати лише 32 ГБ даних під одним ключем. Цей обсяг не був обмеженням під час розробки DES компанією IBM, але в 90-х роках втратив актуальність.

Оскільки AES підтримує розмір блоків у 128 біт, то це дозволяє передавати до 256 ексабайт (254 000 Петабайт) даних без ризику витоку інформації. Оригінальний алгоритм Rijndael підтримує розміри блоків і ключів у 128, 192 та 256 біт, проте в AES блочний розмір завжди становить 128 біт [40].

Алгоритм базується на мережі заміни-перестановки (SP-мережі). Вона складається з послідовних операцій: заміни вхідних даних на конкретні вихідні значення та перестановки бітів.

AES умовно можна поділити на 3 види, кожен з яких буде залежати від довжини ключа:

- AES-128. Для шифрування використовує 128-бітовий ключ шифрування, що забезпечує 10 раундів шифрування. Для ключа існує понад 3,4 квадрильйона потенційних комбінацій.
- AES-192. Для шифрування використовує 192-бітовий ключ, що збільшує кількість раундів шифрування до 12. Така довжина ключа збільшує кількість можливих ключів до 6,2 секстильйона комбінацій.
- AES-256. Для шифрування використовує 256-бітовий ключ, забезпечуючи 14 раундів шифрування та приблизно 1,1 септильйона можливих комбінацій ключів.

У таблиці 2.1 наведені єдині можливі комбінації кількості слів у ключі, розмірів блоку та кількості раундів для AES з різною довжиною ключа.

Таблиця 2.1

Комбінації «Ключ-Блок-Раунд»

	Кількість слів у ключі, Nk	**Розмір блоку, Nb**	**Кількість раундів, Nr**
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Розібравшись в більш загальному стосовно алгоритму AES, можна перейти до опису процесу шифрування цим шифром. Розглядатиметься саме приклад шифрування з ключем у 128 біт.

Перед тим як почати всі вхідні дані розбиваються на блоки по 16 байт, якщо повний розмір не кратний 16 байтам, то дані доповнюється до розміру, кратного 16 байтам. Блоки подаються у вигляді матриці 4x4, яка маю назву state-матриця. Далі виконується додавання до state-матриці основного ключа, відбувається розширення основного ключа (KeyExpansion) і після чого виконуються наступні дії протягом 9 раундів (для ключа довжиною 128 біт всього 10 раундів, але 10 –

фінальний раунд трохи відрізняється від попередніх) [41]. Перелік дій, що виконуються в кожному раунді:

- SubBytes (заміна байтів).
- ShiftRows (зсув рядків).
- MixColumns (перемішування стовпців).
- AddRoundKey (додавання раунд-ключа).

У фінальному раунді виконуються наступні дії:

- SubBytes (заміна байтів).
- ShiftRows (зсув рядків).
- AddRoundKey (додавання раундового ключа).

Тепер більш детально про кожну з цих дій.

Першою було б варто розглянути KeyExpansion – функція, що відповідає за розширення основного ключа для створення раунд-ключів. Розширений ключ складається з 44 слів, з яких кожне по 4 байти (W_i): 4 слова на основний ключ і по 4 слова на 10 раунд-ключів. Таким чином, повна довжина розширеного ключа становить 1408 біт.

Сам процес розширення відбувається наступним чином:

Спочатку 128 бітний ключ ділиться на 4 слова, кожне по 4 байти (32 біти). Нехай наприклад вони будуть мати назви W_0, W_1, W_2, W_3 (рис. 2.11).

Далі для генерації першого слова з наступної четвірки потрібно виконати циклічний зсув попереднього слова, тобто, наприклад, [x_0, x_1, x_2, x_3] стає [x_1, x_2, x_3, x_0]. Після чого кожен байт слова підставляється з таблиці підстановки (S-box), тобто виконується операція SubBytes, яку більш детально буде розібрано пізніше [41].

Рисунок 2.11 – Поділений на слова основний ключ.

Після цього відбувається додавання (XOR) отриманого щойно результату зі словом W_{i-Nk} (Nk – кількість слів у ключі). Потім до першого байту результату цієї дії потрібно додати (XOR) константу раунду $Rcon_i$ [41].

Загальним результатом всіх цих дій і буде утворення першого слова раунд-ключа (рис. 2.12).

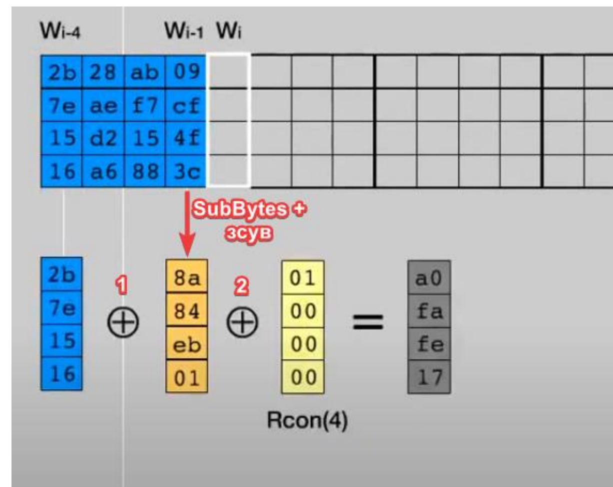


Рисунок 2.12 – Утворення першого слова раунд-ключа.

$Rcon_i$ – це фіксоване значення, яке залежить від номера раунду (i). Воно потрібне, щоб уникнути симетрії в розширених ключах. Воно містить значення $[x^{i-1}$,{00},{00},{00},{00}], де x^{i-1} є степенями x у полі GF(2^8) (коефіцієнти відповідного полінома), а також (i) починається з 1, а не з 0 (рис. 2.13) [41].

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Rcon

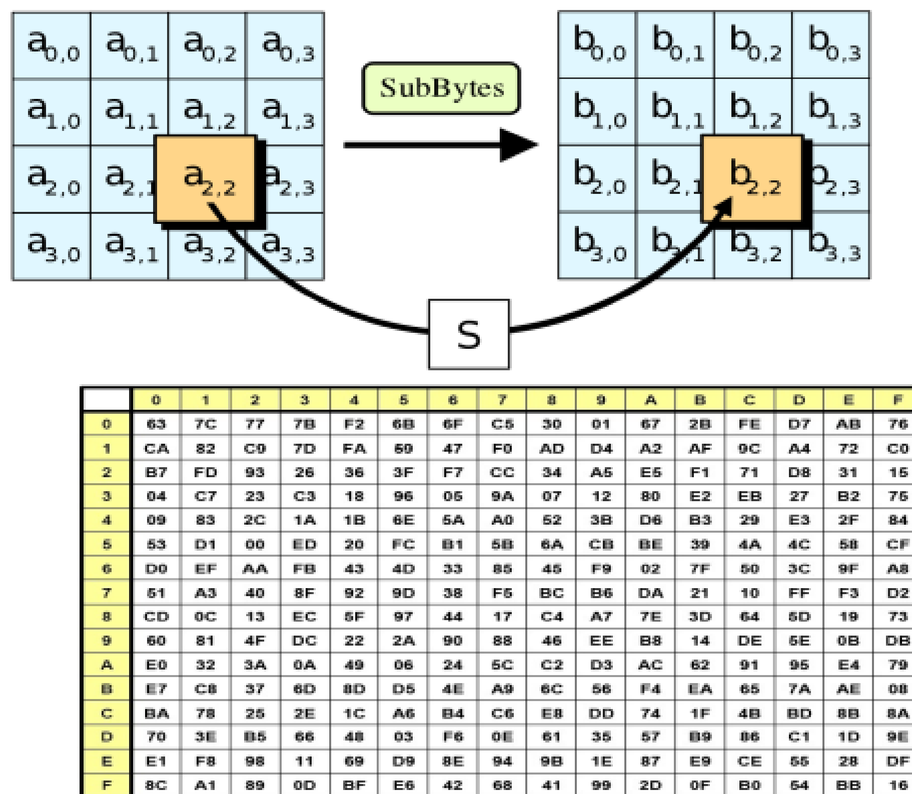
Рисунок 2.13 – Значення $Rcon_i$ для 10 раундів.

Наступні слова раунд-ключа генеруються набагато легше. $W_i = W_{i-1} \oplus W_{i-Nk}$. Для більшого розуміння: $W_5 = W_4 \oplus W_1$, $W_6 = W_5 \oplus W_2$, $W_7 = W_6 \oplus W_3$.

Кожні 4 згенерованих слова утворюють раунд-ключ. Наведені вище дії потрібно повторювати, поки не буде згенеровані всі необхідні раунд-ключі.

Тепер, розібравшись яким чином ключ розширюється, потрібно розібратися, що відбувається під час заміни байтів (SubBytes) в кожному раунді. SubBytes – це функція заміни байтів матриці стану (state). Для цього використовується фіксована таблиця підстановки, яка називається S-box (рис. 2.14). S-box – це матриця 16x16 (256 елементів), яка містить перестановку всіх можливих значень байта (від 0x00 до 0xFF) [41]. Ця таблиця є нелінійною і ретельно розроблена, щоб забезпечити стійкість шифру до різних атак.

Тому, наприклад, щоб виконати SubBytes для байту стану зі значенням 0x9C потрібно знайти значення на перетині рядка 0x9 та стовпця 0xC, тобто після виконання SubBytes значення змінюється на 0xDE.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Рисунок 2.14 – Функція SubBytes та таблиця підстановки S-box.

Далі в кожному раунді виконується функція ShiftRows – циклічний зсув рядків матриці стану (state). Нульовий рядок залишається на місці, перший зміщується вліво на 1 байт, другий на 2 байти і третій на 3 відповідно (рис. 2.15) [41].

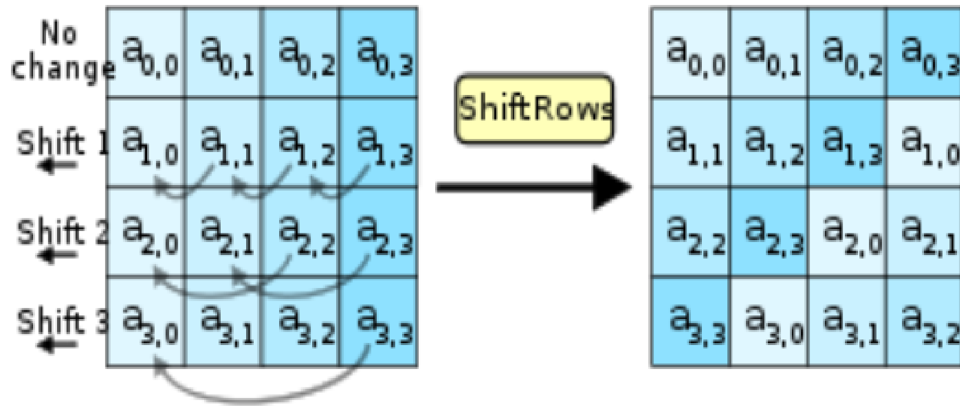


Рисунок 2.15 – Функція ShiftRows.

Після здійснення зсуву рядків в алгоритмі AES виконується функція MixColumns – множення кожного стовпця матриці стану на фіксовану матрицю. Таким чином здійснюється лінійне перетворення над стовпцями матриці стану (рис. 2.16).

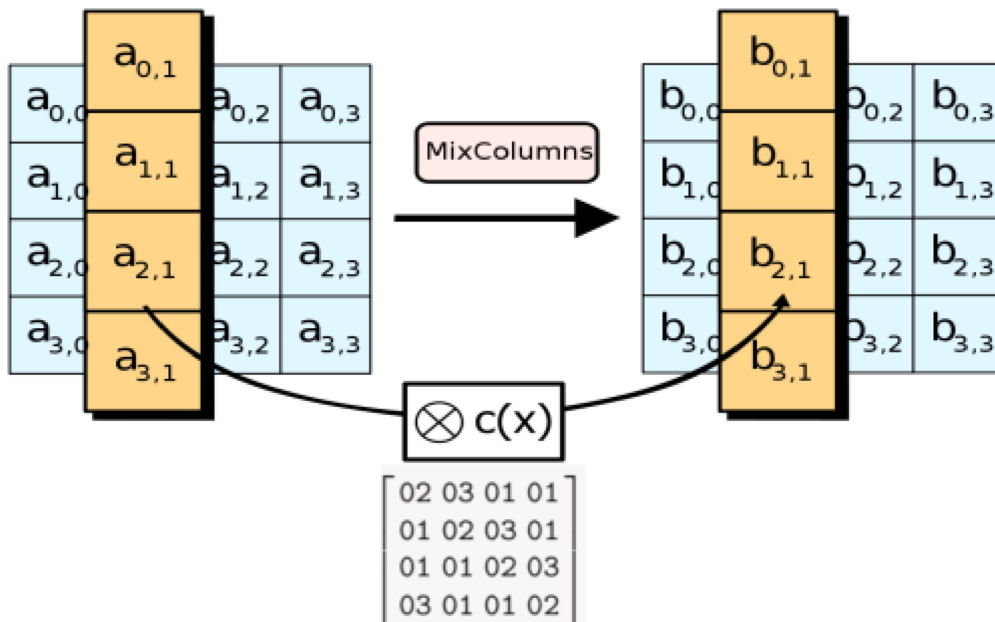


Рисунок 2.16 – Функція MixColumns.

В кінці кожного раунду до матриці стану елемент за елементом додається (XOR) значення раунд ключа K_i, де i – номер раунду (рис. 2.17).

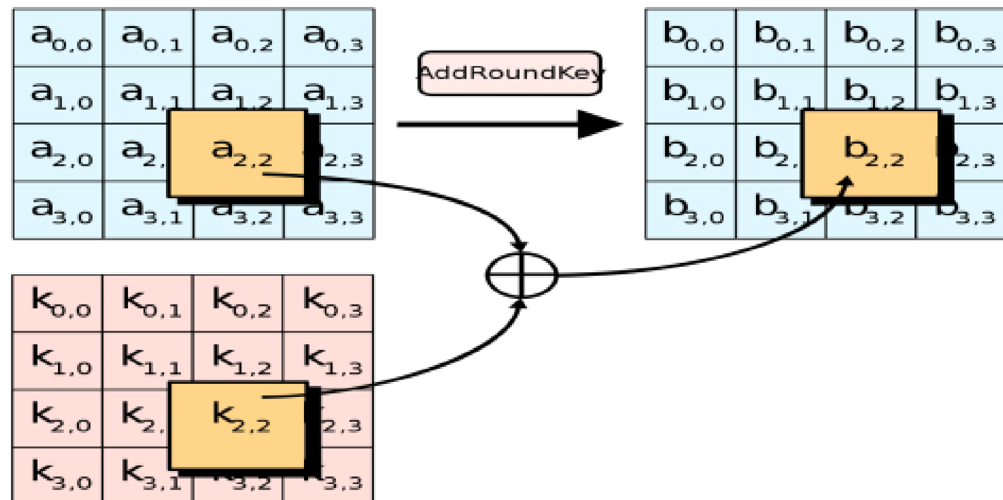


Рисунок 2.17 – Функція AddRoundKey.

Процес дешифрування являє собою послідовність інвертованих операцій шифрування, що виконуються у зворотному порядку:

Спочатку відбувається розширення ключа.

Потім виконується 9 раундів дешифрування, кожен з яких складається з перетворень:

- AddRoundKey – додавання (XOR) матриці стану з раунд-ключем.
- InverseMixColumns – зворотна перестановка стовпців матриці стану.
- InverseShiftRows – зворотний циклічний зсув стовпців матриці стану.
- SubBytes – заміна байтів матриці стану за зворотною таблицею замін InverseS-box.

Фінальний раунд знову трохи відрізняється, бо у ньому виконується лише наступні перетворення:

- AddRoundKey
- InverseShiftRows
- InverseSubBytes

В залежності від того, яким чином застосовуються симетричне блочне шифрування або дешифрування, AES може бути реалізований з використанням різних режимів роботи [42]. Вибір самої реалізації повністю залежить від

конкретних вимог програми, де відбувається ця реалізація, та бажаних властивостей безпеки. Найпоширенішими вважаються наступні режими роботи:

1. ECB (Electronic Codebook). Розбиває відкритий текст на блоки; кожен блок шифрується незалежно за допомогою алгоритму AES (рис. 2.18). Цей режим простий і його можна виконувати паралельно, але він не підходить для шифрування великих обсягів даних або даних, що повторюються, оскільки може призвести до появи шаблонів у зашифрованому тексті.

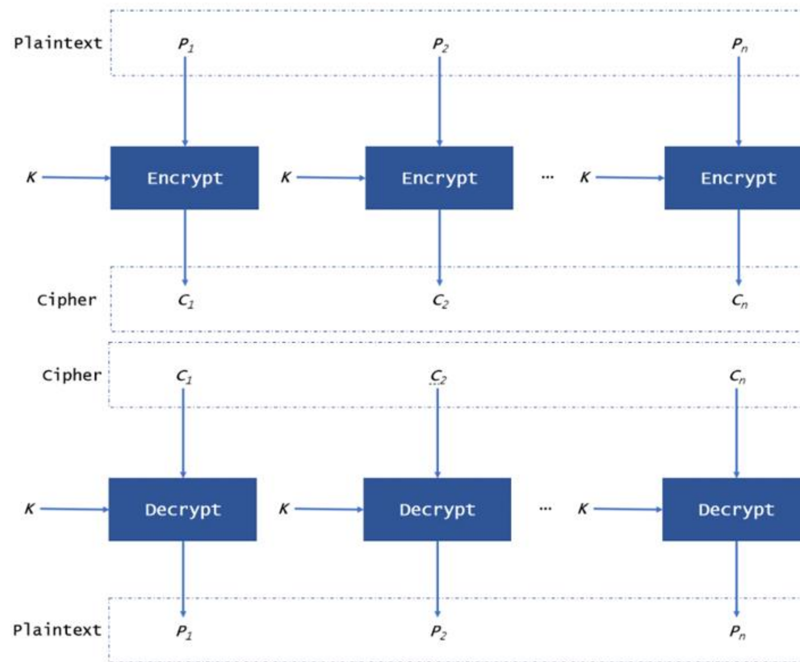


Рисунок 2.18 – Зображення режиму роботи ECB.

2. CBC (Cipher Block Chaining). Перед шифруванням попередній блок зашифрованого тексту додається (XOR) до блоку відкритого тексту (рис. 2.19). Це вводить залежність між блоками, що робить його більш захищеним, ніж ECB, проти атак на основі аналізу шаблонів. Вектор ініціалізації (IV) використовується для шифрування першого блоку, а зашифрований текст кожного блоку використовується як IV для наступного блоку.

3. CFB (Cipher Feedback). Перед початком шифрується IV, потім виконується додавання (XOR) з блоком відкритого тексту, щоб отримати зашифрований текст. Після чого результат шифрування знову шифрується і вже потім додається (XOR) до відкритого тексту (рис. 2.20). Цей режим не шифрує

відкритий текст безпосередньо, він використовує зашифрований текст та додає (XOR) його до відкритого тексту, щоб отримати новий зашифрований текст. Також у цьому режимі не є обов'язковим використання доповнення (padding), бо у цьому режимі AES з блочного шифру перетворюється на потоковий, для якого не обов'язкові повні блоки.

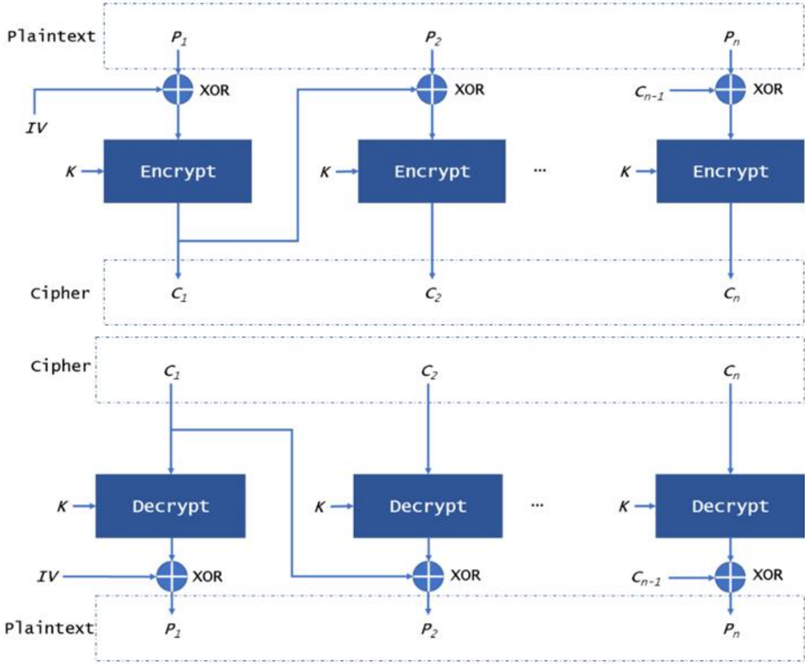


Рисунок 2.19 – Зображення режиму роботи CBC.

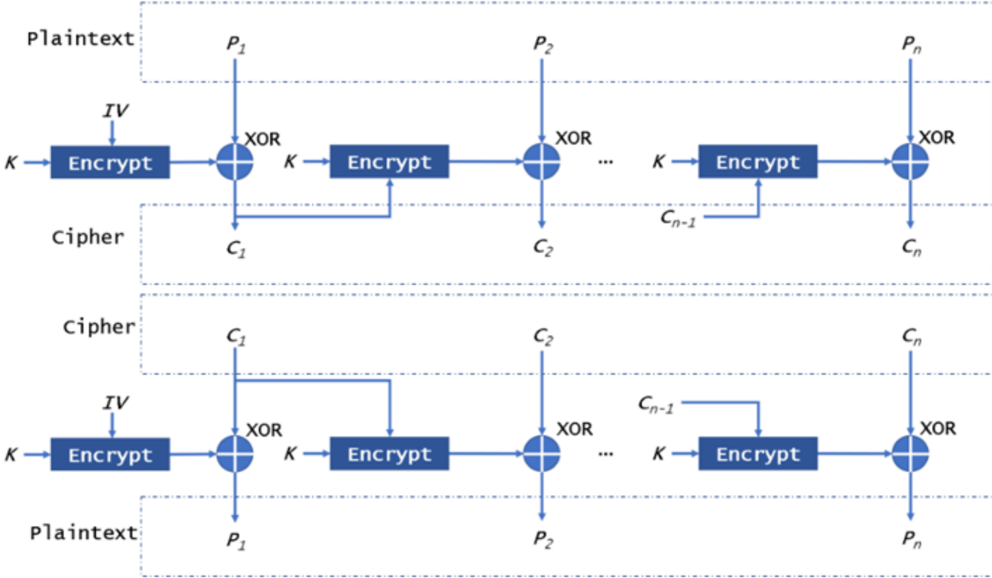


Рисунок 2.20 – Зображення режиму роботи CFB.

4. OFB (Output Feedback). У цьому режимі спочатку шифрується IV. Потім виконується додавання (XOR) зашифрованого IV з відкритим текстом, щоб отримати зашифрований результат. Цей IV переходить у наступний етап, знову шифрується і додається до відкритого тексту для отримання наступної частини зашифрованого тексту (рис. 2.21). OFB вимагає послідовного шифрування і розшифрування, оскільки помилки при передачі можуть поширюватися на наступні блоки.

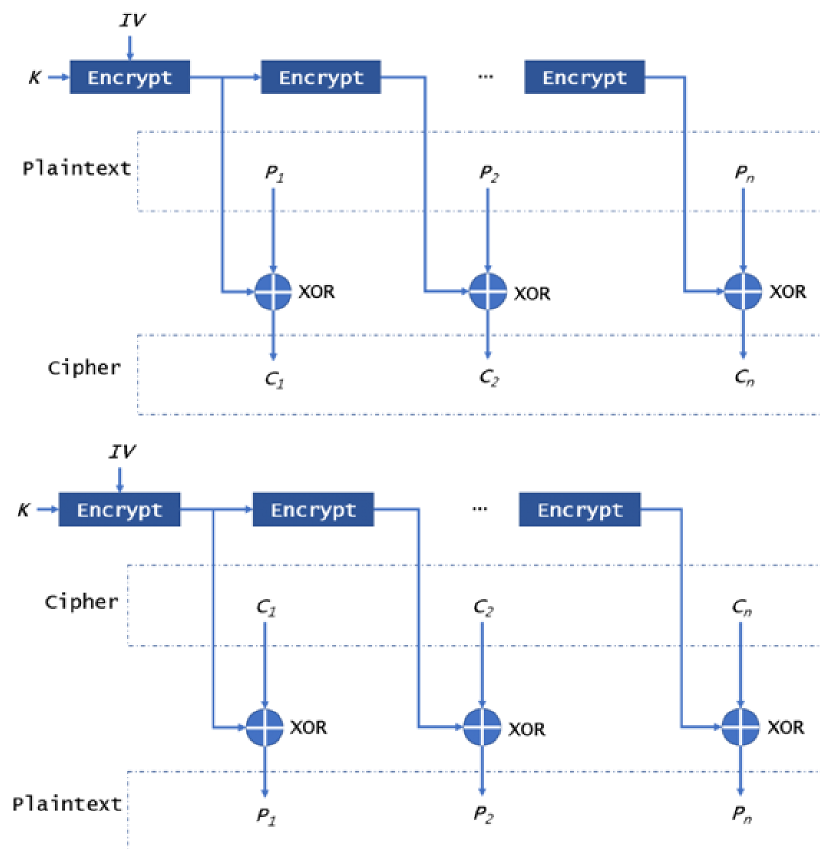


Рисунок 2.21 – Зображення режиму роботи OFB.

5. CTR (Counter). Використовує значення лічильника в поєднанні з унікальним nonce (унікальним значенням, яке використовується лише один раз для даного ключа) як вхідні дані. Після цього шифрує їх і для отримання зашифрованого тексту вони додаються до відкритого тексту (рис. 2.22). Режим CTR дозволяє паралельне шифрування і розшифрування.

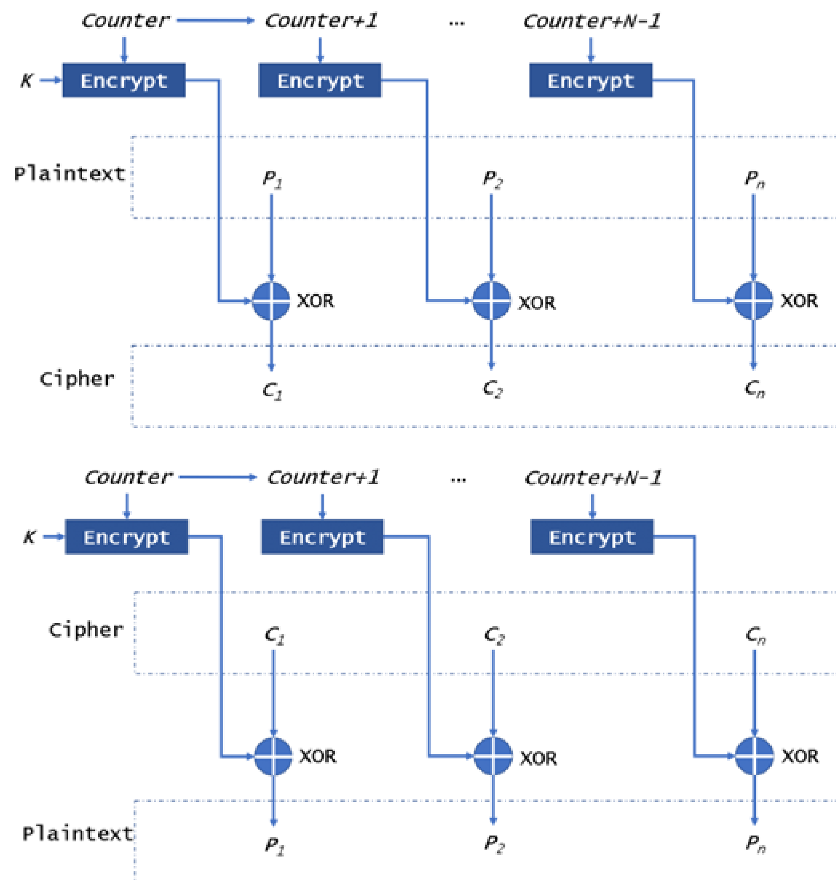


Рисунок 2.22 – Зображення режиму роботи CTR.

2.2.3 Алгоритм Bcrypt

Bcrypt – це функція хешування паролів, розроблена Нільсом Провосом та Девідом Мазьєресом у 1999 році, яка стала популярною як алгоритм хешування паролів за замовчуванням для OpenBSD. Порівнюючи з більш простими функціями хешування, то головною перевагою використання bcrypt є те, що з’являється можливість встановити, наскільки дорого буде обчислюватися хеш. Це називається розтягуванням ключа і може використовуватися з будь-яким паролем, наданим користувачем, для захисту від атак грубої сили.

Bcrypt у своїй основі для розтягування ключа використовує симетричний блочний шифр Blowfish, який за принципом мережі Фейстеля для шифрування інформації (рис. 2.23).

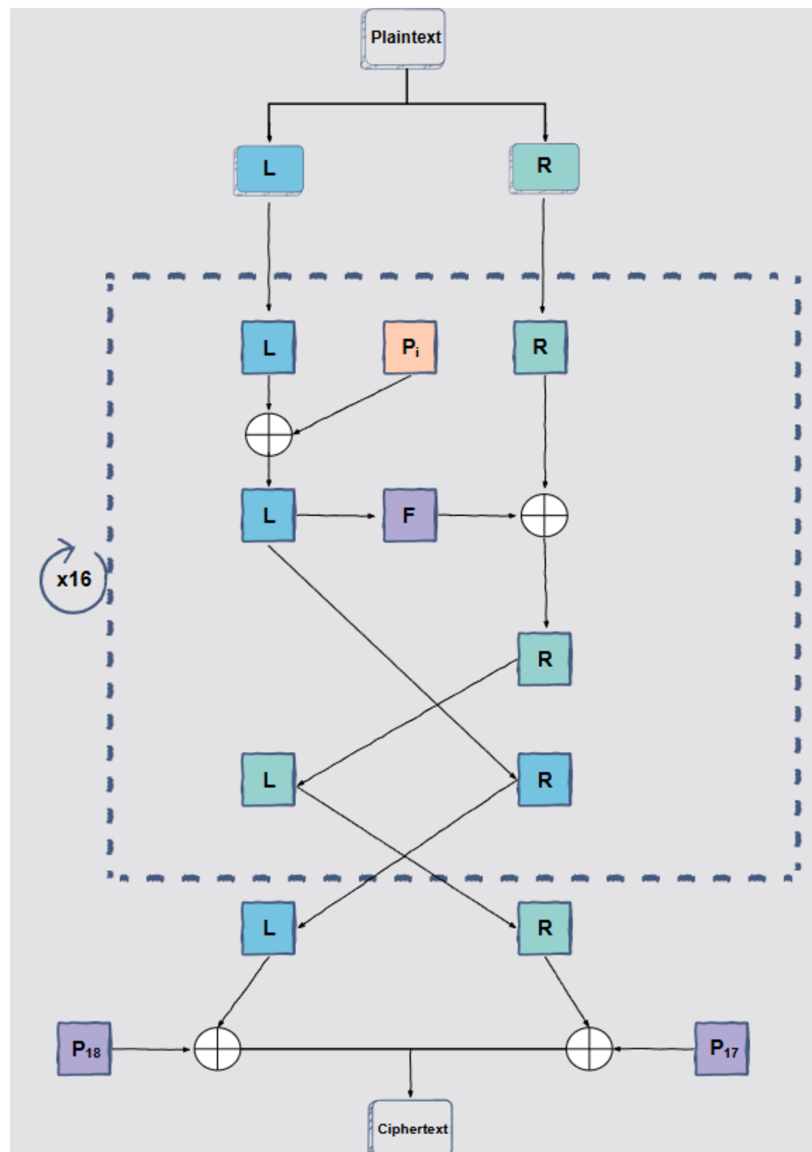


Рисунок 2.23 – Загальний вигляд алгоритму Blowfish.

Головною відмінністю Bcrypt від Blowfish є спеціально модифікована функція розкладання ключів – Eksblowfish. Eksblowfish – це дороговартісна варіація блочного шифру Blowfish з розкладом ключів Blowfish. Етапи виконання цієї функції наведено нижче [25]:

1. Спочатку вона приймає пароль (масив байтів розміром 1-72 байт), сіль (випадково згенерований масив байтів довжиною 16 байт) та вартість (число від 4 до 31, яке вказує кількість ітерацій у вигляді 2^{cost}) як вхідні дані.

2. Виконується заповнення P-таблиці (підключі) та S-box шляхом додавання до них цифр числа π (у шістнадцятковому форматі).

3. Викликається функція ExpandKey, яка модифікує початкові таблиці P та S, перемішуючи їх за допомогою пароля і солі.

4. Наступним кроком виконується 2^{cost} циклів, у кожному з яких таблиці P та S знову переміщуються, спочатку з паролем, потім із сіллю.

5. В самому кінці повертає перемішані таблиці P та S як ключі, які будуть використовуватися для подальшого шифрування.

Для модифікації початкових значень таблиць підключів (P) та S-блоків, як вже було сказано, використовується функція ExpandKey, алгоритм дій якої наступний:

1. Приймає значення паролю, солі, початкових таблиць P та S.

2. Змішує пароль з початковою таблицею P. Для цього послідовно для всіх 18 елементів P виконується операція додавання (XOR) із 32 байтами пароля (якщо пароль коротший за 32×18 байтів, то він циклічно повторюється).

3. 128-бітна сіль розділяється на дві частини по 64 біти та генерується 8-байтовий (64-бітовий) блок, з початковим значенням 0.

4. Протягом 9 ітерацій у блок почергово підмішується обидві частини солі, після чого виконується шифрування цього блоку з використанням поточного стану (P і S). Потім оновлюються елементи P-таблиці результатом цього шифрування. Таким чином замінюються по два ключа за ітерацію.

5. Далі теж саме виконується і до таблиць S. Спочатку у блок додаються (XOR) почергово обидві частини солі, потім він шифрується з використанням поточних станів таблиць, після чого перезаписуються по 2 слова у S таблиці.

6. В кінці ця функція повертає повністю модифіковані (перемішані) таблиці S та P, які далі будуть основою для наступних кроків.

Після отримання набору підключів та S-блоків від Eksblowfish, bcrypt шифрує пароль повторно 64 рази в режимі ECB (Electronic Codebook). Результат шифрування потім об'єднується з параметрами cost та salt та кодуються за допомогою Base64 перед тим як отримати остаточний результат. Загальний вигляд хешу bcrypt має наступний вигляд:

$2<a/b/x/y>$[вартість]$[22 символи солі][31символ хешу]

Тобто, наприклад, візьмемо пароль qwerty1 та використаємо bcrypt для його хешування, в результаті отримаємо значення:

$2a$12$K9vuU0Shg1LH/jCt22T8bujjCIOdEd7/.miRGQx9kjAaHztjccaWG,

де 2a – це версія bcrypt, 12 – це вартість, тобто кількість раундів при розкладанні ключа (2^{12}), значення солі має значення – K9vuU0Shg1LH/jCt22T8bu, а сам хеш паролю – jjCIOdEd7/.miRGQx9kjAaHztjccaWG. Схема алгоритму bcrypt наведена на рисунку 2.24.

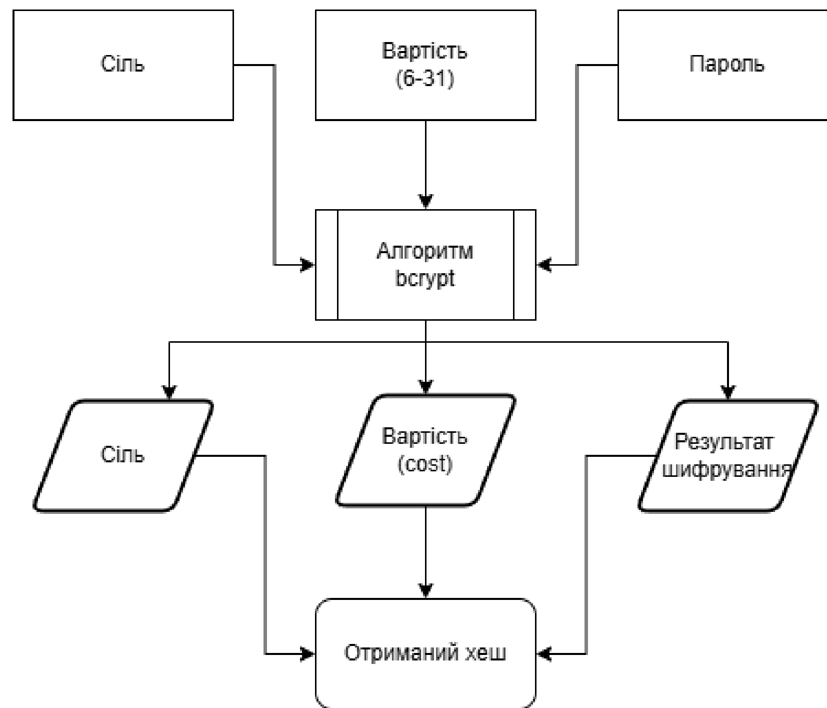


Рисунок 2.24 – Загальний вигляд алгоритму bcrypt.

Відповідно до [25] bcrypt може приймати вхідні дані максимальною довжиною до 72 байт, тобто якщо буде введено більше цього значення, то все інше, що поза цими 72 байтами буде обрізано. Таке обмеження є однією з найвагоміших проблем bcrypt, бо воно може погано впливати на процес автентифікації користувача.

2.3 Обґрунтування вибору алгоритмів для розробленого методу

Надійне зберігання паролів у базах даних є головними аспектом безпеки для будь-яких веб-застосунків. Неправильна реалізація цього питання

призводить до можливого витоку конфіденційних даних або пошкодження всієї системи. Для вирішення даної проблеми потрібно вибирати оптимальне рішення, бо захист паролів під час їх зберігання є критично важливим.

У даній роботі для зберігання паролів було обрано комбінацію алгоритмів AES-256 та SHA-512, де паролі спочатку шифруються за допомогою AES-256, а потім отримані зашифровані дані хешуються з використанням SHA-512.

AES є одним із найпоширеніших і найнадійніших алгоритмів симетричного шифрування. Вибрана версії алгоритму шифрування використовує 256-бітний ключ (32 символи), забезпечує високу стійкість до атак методом грубої сили або інших відомих криптоаналітичних атак. Використання сучасних обчислювальних потужностей не дає гарантій на злам шифру AES-256, бо він досі вважається майже неможливим. AES має дуже велику швидкість шифрування, що для веб-застосунків, де присутній процес автентифікації, є важливим фактором. Наступною перевагою AES, як методу шифрування, є його широка підтримка програмними та апаратними рішеннями, а також те, що він є стандартом для захисту даних у США. Також використання шифрування, дає теоретичну можливість відновити оригінальний пароль, наприклад, у ситуаціях коли потрібно надіслати користувачеві тимчасову копію паролю або забезпечити синхронізацію з іншими системами, що не приймають хеші.

Після шифрування, дані обробляються за допомогою хеш-функції SHA-512, яка генерує 512-бітний хеш (128 символів), і розроблена таким чином, щоб мінімізувати ймовірність колізій – ситуацій, коли різні вхідні дані призводять до однакового хешу. Це робить практично неможливим підбір оригінального (зашифрованого) пароля за його хешем. У даному випадку SHA-512 має додаткову стійкість до атак з використанням "райдужних таблиць", тому що перед хешуванням паролі шифруються, що і унеможливлює зловмисника скористатися стандартними "райдужними таблицями", бо для цього йому доведеться створювати таблиці для кожного унікального ключа шифрування.

Не зважаючи на велику швидкість підбору хешу, SHA-512 все одно залишається досить стійкою до цієї атаки, бо виконавши хешування паролю з 8

символів великих/малих літер, чисел та хоча б одного спеціального символу, 8 графічних адаптерів Nvidia GTX 1080 за допомогою програми hashcat потрібно більше 10 років, щоб підібрати такий пароль (рис. 2.25).

	Number of characters							
Cracking speed: ~1160 kH/s	4	5	6	7	8	9	10	11
Default hashcat mask (?1?2?2?2?2?2?2?3?3?3?3)	4s	1min 30sec	54min	1d8h	48d 13h	4y 287d	>10y	>10y
Full charset (?a?a?a?a?a?a?a?a?a?a?a)	1min	1h51 min	7d4h	1y 331d	>10y	>10y	>10y	>10y

Рисунок 2.25 – Підбір хешу SHA-512 для паролів.

У свою чергу, алгоритм bcrypt часто рекомендується як один з найкращих алгоритмів для хешування паролів, бо забезпечує високий рівень безпеки. Це досягається завдяки адаптивності алгоритму (можливість вибирати вартість). Використання обраних (AES-256 та SHA-512) алгоритмів разом надає більшу гнучкість. Наприклад, якщо в майбутньому буде необхідним реалізація додаткових механізмів безпеки, що вимагають доступу до оригінального пароля (наприклад, двофакторна автентифікація з одноразовими паролями, які генеруються на основі основного пароля), то AES-256 надає можливість це зробити, в той час як bcrypt є незворотнім. Швидкість обробки пароля з використанням AES-256 та SHA-512 також буде вищою, ніж у bcrypt. Використання обраної комбінації алгоритмів надає повний контроль над процесом шифрування та хешування, що дозволяє більш точно налаштувати параметри безпеки та інтегрувати цей процес у загальну архітектуру безпеки застосунку.

Послідовність шифрування → хешування – є важливою для забезпечення безпеки. Такий підхід робить неможливим застосування стандартних атак на основі "райдужних таблиць" без знання ключа шифрування. Навіть у

малоймовірному випадку зламу хеш-функції, зловмисник отримає лише зашифрований пароль, який не зможе використати без ключа AES-256.

2.4 Висновки до другого розділу

У цьому розділі було проведено детальний аналіз алгоритмів захисту паролів, які застосовуються у веб-застосунках. Було розглянуто фундаментальні принципи використання хеш-функцій, симетричного й асиметричного шифрування.

Прості алгоритми швидкого хешування (SHA-256 чи MD5) мають недостатню захищеність для чутливої інформації в базах даних від спроб використання багатьох сучасних атак. Для покращення стійкості таких алгоритмів бажано використовувати додаткові засоби захисту. Алгоритми Bcrypt або Argon2 можуть компенсувати таку вразливість, збільшуючи необхідність великої кількості часу, але також вони потребують набагато більше системних ресурсів під час їх виконання.

Практика використання симетричного шифрування за допомогою AES-256 вже багато років доводить його здатність забезпечувати високий рівень надійності зберігання інформації, але лише за умови належного та безпечного управління ключами шифрування. Також в ході аналізу було зазначено, що IV та відповідні режими блочного шифрування (CBC, CFB, CTR тощо) значно знижують ризик утворення повторюваних шаблонів у зашифрованому тексті, що підвищує стійкість до криптоаналізу.

РОЗДІЛ 3
РОЗРОБКА МЕТОДУ ЗБЕРІГАННЯ ПАРОЛІВ У ВЕБ-ЗАСТОСУНКАХ

3.1 Засоби реалізації розробленого веб-застосунку

JavaScript

JavaScript - це мова програмування, яку використовується розробниками для створення інтерактивних веб-сторінок. Функції JavaScript можуть поліпшити зручність взаємодії користувача з веб-сайтом: від оновлення стрічки новин у соціальних мережах і до відображення анімації та інтерактивних карт. JavaScript є мовою програмування під час розроблення скриптів для виконання на стороні клієнта, що робить його однією з базових технологій у всесвітній мережі Інтернет. Наприклад, карусель зображень, меню елементів, що випадають за натисканням кнопки миші, і динамічно мінливі кольори елементів на веб-сторінці, це все може бути виконано за допомогою JavaScript [43].

JavaScript найчастіше використовується у поєднанні разом з HTML та CSS. Ці три технології вважаються стандартними для веб-розробки та веб-технологій загалом. Якщо спробувати легко описати призначення цих стандартів будь-якої веб-розробки, то отримаємо наступне:

HTML – це мова розмітки, яка використовується для структурування та надання сенсу веб-контенту, наприклад, визначення абзаців, заголовків і таблиць даних або вбудовування зображень і відео на сторінку.

CSS – це мова правил стилів, яка використовується для застосування стилів до створеного HTML-контенту, наприклад, встановлення кольорів фону та шрифтів, а також розміщення самого контенту в декілька колонок.

JavaScript – це мова сценаріїв, яка дозволяє створювати динамічно оновлюваний контент, керувати мультимедіа, анімувати зображення і робити майже все інше.

Також можна навести просту аналогії для розуміння взаємодії між цими технологіями: HTML можна уявити як структура будинку, його каркас, тоді CSS матиме роль внутрішнього та зовнішнього декору, а JavaScript буде функціональними можливостями, які роблять будинок придатним для життя (наприклад система електроживлення, водопостачання та інші).

JavaScript використовується для:

- Збереження значення всередині змінних для подальшої роботи з цими даними (наприклад збереження імені користувача з поля вводу і тп).
- Виконання іншого коду у відповідь на певні події, що відбуваються на веб-сторінці (наприклад оновлення сторінки після натискання користувачем на відповідну кнопку).
- Створення API (Application Programming Interfaces).
- Розробка мобільних додатків з зручним та інтерактивним користувацьким інтерфейсом.

Найголовнішим призначенням JavaScript є саме розробка API – набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек. API дозволяє приховати внутрішні деталі роботи системи, показуючи лише ті частини, які можуть бути корисними іншим розробникам. Таким чином, API-інтерфейси дозволяють компаніям надавати доступ до своїх ресурсів, зберігаючи безпеку та повний контроль (рис. 3.1).

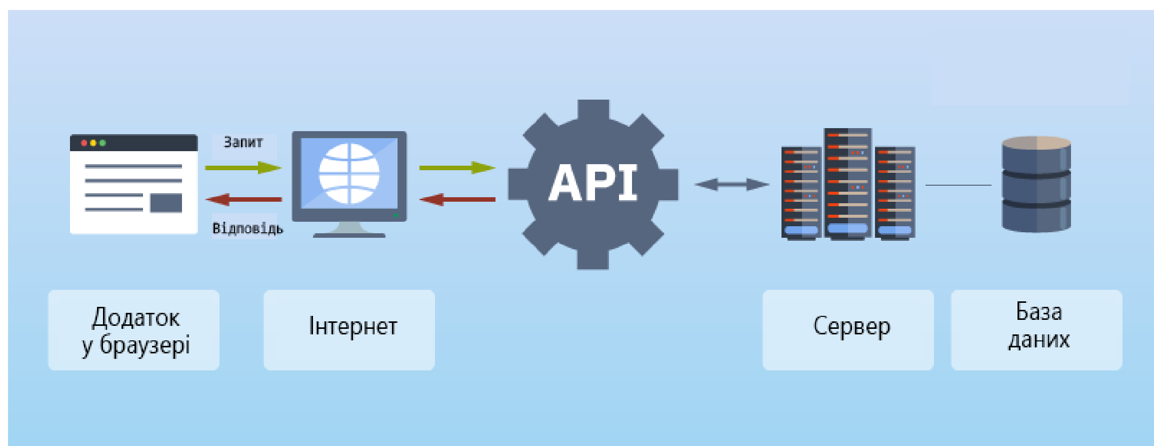


Рисунок 3.1 – Загальний вигляд застосування API.

Серед переваг використання JavaScript можна виділити наступні:

- Цю мову програмування легко вивчати та використовувати. Її синтаксис на відміну від інших мов програмування не є залежною від системи на якій може запуститися, тому виконується вона всередині веб-браузера.
- JavaScript може використовуватися для зменшення навантаження на сервер і перевантаження мережі, оскільки може виконувати логічні операції і багато роботи іншої нескладної роботи сервера на самому клієнті (наприклад перевірка на правильність введення даних при реєстрації).
- JavaScript покращує користувацький інтерфейс, розширяючи функціональність і читабельність, а також роблячи взаємодію з користувачем сайту більш ефективною.

Node.js

Node.js – це середовище виконання JavaScript, яке дає змогу запускати код JavaScript на стороні сервера. Воно використовує рушій V8 від Google для виконання JavaScript-коду, що дає змогу виконувати операції вводу/виводу асинхронно й ефективно, працювати з мережею та файловою системою. Написані на Node.js додатки призначені для використання як веб-сервери, фреймворки, інструменти для резервного копіювання та багато інших проектів. Node.js також дає змогу використовувати пакетний менеджер npm для встановлення та керування залежностями [44].

Node.js використовує архітектуру «Однопотокового циклу обробки подій» для одночасної роботи з декількома клієнтами (рис. 3.2):

1. Коли надходить запит, Node.js поміщає його в чергу.
2. Після цього починає працювати однопотоковий «Цикл обробки подій» – основний компонент, нескінченно очікує на запити.
3. Коли запит з'являється, цикл подій вибирає його з черги та перевіряє, чи потребує він блокуючої операції введення/виведення (відбувається перевірка чи є цей запит простим та швидким до виконання).
4. Якщо ні (тобто операція не блокуюча), він обробляє запит і надсилає відповідь.

5. Якщо запит вимагає виконання блокуючої операції, цикл подій призначає для його обробки один з внутрішніх потоків. Кількість цих внутрішніх потоків обмежена. Ця група допоміжних потоків називається групою виконавців.

6. Цикл подій відстежує виконання блокуючих завдань і, коли завдання завершено, відповідний обробник ставиться в чергу для виконання циклом подій. Після чого цикл обробки подій знову оброблює вже виконане завдання та надсилає відповідь.

Node.js використовує менше потоків, тому він використовує менше ресурсів/пам'яті, що призводить до швидшого виконання завдань. У деяких ситуаціях ця однопотокова архітектура еквівалентна багатопотоковій. Але коли потрібно обробляти завдання з великими обсягами даних, то використання багатопотокових мов програмування є більш кращим вибором. Node.js є найкращим вибором додатків, що працюють в реальному часі.

Node.js часто використовується для створення серверної архітектури багатьох веб-додатків, але найчастіше його використовують для:

- Чати в реальному часі – завдяки своїй однопотоковій асинхронній природі Node.js добре підходить для обробки спілкування в реальному часі. Він легко масштабується і часто використовується для створення чат-ботів.
- Node.js можна використовувати для розроблення серверних компонентів IoT-проектів завдяки своїй ефективності в обробленні великої кількості з'єднань і даних у реальному часі.
- Потокове передавання даних, будь-яких даних – від відео до журналів подій програм.
- Складні односторінкові додатки (Single-page applications – SPA) – у SPA весь додаток завантажується на одній сторінці. Зазвичай це означає, що у фоновому режимі виконується декілька запитів до певних компонентів. Тут на допомогу приходить цикл обробки подій Node.js, який швидко оброблює ці запити.
- Створення гнучких та маштабованих систем керування контентом, завдяки можливості змінювати дані в реальному часі.

- При розробці мікросервісної архітектури, де кожен сервіс може реалізовуватися як окремий застосунок, який звертається до інших сервісів за допомогою API.

Express.js

Express.js — це мінімалістичний, швидкий серверний фреймворк для Node.js, який дає можливість розробникам використовувати функції та інструменти для розробки масштабованих серверних програм з чіткою та зрозумілою структурою [45].

Фреймворк надає набір інструментів та готових рішень для роботи з HTTP-запитами та відповідями, маршрутизацією, проміжним програмним, що забезпечує створення та розгортання великих корпоративних додатків.

Як раніше описувалося в роботі, Node.js – це середовище виконання, яке дозволяє запускати JavaScript на стороні сервера. Він не надає жодних спеціальних механізмів створення та взаємодії з веб-застосунками. В свою чергу Express.js забезпечує рівень абстракції над Node.js, спрощуючи взаємодію з такими завданнями, як маршрутизація, керування сесіями та обробка HTTP-запитів.

Express.js створений та призначений для керування основними функціями веб-сервера, дозволяючи розробникам зосередитися на логіці та взаємодії функцій розробленого ними веб-додатку.

Express.js використовує клієнт-серверну модель для прийому запитів користувачів і надсилання відповідей клієнту. Тому нижче наведено процес того, як працює Express.js:

1. Коли користувач надсилає запит зі свого веб-браузера, вводячи адресу веб-сайту, браузер надсилає HTTP-запит серверу.

2. Сервер отримує запит по одному зі своїх маршрутів і обробить його, використовуючи контролер, який відповідає запитуваному маршруту.

3. Після обробки сервер надсилає відповідь клієнту, використовуючи протокол HTTP, оскільки він може використовуватися для двостороннього зв'язку.

4. Відповідь, що повертається клієнту, може бути стандартним текстом, динамічною HTML-сторінкою, яку браузер обробить і відобразить красиву веб-сторінку, або JSON-даними, які оброблятимуть фронтенд-розробники для відображення інформації на веб-сторінці.

MySql

MySQL – це система управління базами даних з відкритим вихідним кодом, яка використовує SQL для створення та керування базами даних. Акронім «SQL» означає мову структурованих запитів – тип мови програмування, який використовується для маніпулювання даними в базі даних. Оскільки MySQL використовує мову структурованих запитів (SQL) для керування даними у базах даних тому вона і використовує цей акронім як частину своєї назви. Також маючи дані, які зберігаються в MySQL, СУБД підтримує можливість використання простих підказок оригінальної SQL для додавання, пошуку, аналізу та отримання цих даних [46]. Мова запитів SQL вважається найпоширенішою мовою програмування, яка використовується для доступу до баз даних.

MySQL є реляційною базою даних і тому зберігає їх в таблицях рядків і стовпців, організованих у схеми. Схема визначає, як дані організовані та зберігаються, а також описує зв'язок між різними таблицями. За допомогою такого формату розробники мають можливість легко зберігати, отримувати та аналізувати багато типів даних, таких як простий текст, числа, дати, час, а також нещодавно додали підтримку JSON-файлів і векторів.

Структура бази даних організована у файли, оптимізовані для швидкого доступу до даних. Така логічна модель даних із такими об'єктами, як таблиці даних, рядки та стовпці, пропонує розробникам і адміністраторам баз даних гнучке налаштування всіх процесів. Вони можуть встановлювати правила, що регулюють взаємозв'язки між різними полями даних, наприклад один до одного, один до багатьох, унікальні, обов'язкові чи необов'язкові, а також додавати «вказівники» між різними таблицями. СУБД MySql забезпечує дотримання цих правил, щоб за допомогою добре спроектованої бази даних програма ніколи не бачила даних, які є неправильними, дубльованими або застарілими.

База даних MySQL — це клієнт/серверна система, яка складається з багатопотокового SQL-сервера, який підтримує різні серверні частини, кілька клієнтських програм і бібліотек, набір інструментів адміністрування та широкий вибір інтерфейсів прикладного програмування (API). MySQL доступна як вбудована багатопотокова бібліотека, яку розробники можуть пов'язувати з додатками, щоб отримати менший, швидший і простий в управлінні автономний продукт.

3.2 Реалізації алгоритму шифрування та хешування

Основною метою роботи є створення нового методу захисту паролів, що зберігаються в базах даних. Для досягнення цієї мети було обрано спосіб, що вважається відносно легким у реалізації, швидким та надійним – комбінування двох існуючих методів забезпечення цілісності даних: шифрування за допомогою AES-256 CBC та подальшого хешування отриманих зашифрованих даних за допомогою SHA-512. Обидва алгоритми є поширеними, а їхня стійкість до компрометації доведена багаторічним використанням. Тому в розробленому веб-додатку ключовими процесами є саме шифрування та хешування.

Як вже зазначалося, для одного з етапів захисту даних було обрано AES-256 CBC та розроблено власну функцію для шифрування даних (рис. 3.2).

Ця функція на вході приймає значення ключа, дані, що потрібно зашифрувати та IV. Потім функція конвертує надані значення в байти та ініціює процес шифрування відповідно до алгоритму, описаного в попередньому розділі. Тобто, спочатку генерується вектор ініціалізації, щоб кожне зашифроване повідомлення було унікальне, або якщо вже було його надано то використовується він (збереження та перевірка на наявність вектору ініціалізації потрібна для авторизації користувачів). Після цього отримані дані розбиваються на блоки, кожен по 16 байт, і для кожного блоку виконується додавання за модулем 2 (операція XOR) з попереднім блоком, за винятком першого, він додається до вектору ініціалізації. Потім кожен блок шифрується за допомогою

алгоритму AES та в результаті функція повертає вже зашифровані дані та вектор ініціалізації, щоб при створенні облікового запису користувача його можна було зберегти у окрему таблицю бази даних.

```
function aesEncryptCBC(message, keyString, ivProvided) {
    const key = stringToBytes(keyString);
    let bytes = stringToBytes(message);
    padPKCS7(bytes);
    let iv;
    if (ivProvided) {
        iv = ivProvided;
    } else {
        const ivBuffer = crypto.randomBytes(16);
        iv = Array.from(ivBuffer);
    }
    let prev = iv.slice();
    let ciphertext = [];
    for (let i = 0; i < bytes.length; i += 16) {
        const block = bytes.slice(i, i + 16);
        const xored = xorBytes(block, prev);
        const encryptedBlock = aesEncryptBlock(xored, key);
        ciphertext = ciphertext.concat(encryptedBlock);
        prev = encryptedBlock;
    }
    return { iv, ciphertext };
}
```

Рисунок 3.2 – Функція шифрування даних за допомогою AES-256 CBC.

Функція шифрування окремого блоку даних зображена на рисунку 3.3. Вона приймає блок даних для шифрування та ключ. Потім виконується процедура розширення ключа, і протягом 13 раундів відбуваються такі операції: заміна байтів блоку згідно з S-таблицею, зсув рядків, перемішування стовпців та додавання раундового ключа. У фінальному, 14-му раунді, виконуються ті самі операції, окрім перемішування стовпців. Результатом роботи функції є зашифрований блок.

```
function aesEncryptBlock(input, key) {
    let state = input.slice();
    const expandedKey = keyExpansion(key);
    const Nr = 14;
    addRoundKey(state, getRoundKey(expandedKey, 0));
    for (let round = 1; round < Nr; round++) {
        subBytes(state);
        shiftRows(state);
        mixColumns(state);
        addRoundKey(state, getRoundKey(expandedKey, round));
    }
    subBytes(state);
    shiftRows(state);
    addRoundKey(state, getRoundKey(expandedKey, Nr));
    return state;
}
```

Рисунок 3.3 – Функція шифрування блоків AES.

Функція розширення ключа є критично важливою для AES, оскільки вона генерує раундові ключі, необхідні для кожного етапу шифрування. Без коректного розширення ключа процес шифрування буде недійсним. Тому цю функцію було розроблено відповідно до специфікації AES-256 (рис. 3.4).

```
function keyExpansion(key) {
    const Nk = 8;
    const Nb = 4;
    const Nr = 14;
    const w = [];
    let temp = [];

    for (let i = 0; i < Nk; i++) {
        w[i] = [key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i + 3]];
    }
    for (let i = Nk; i < Nb * (Nr + 1); i++) {
        temp = w[i - 1].slice();
        if (i % Nk === 0) {
            temp = subWord(rotWord(temp));
            temp[0] ^= Rcon[i / Nk];
        } else if (i % Nk === 4) {
            temp = subWord(temp);
        }
        w[i] = [];
        for (let j = 0; j < 4; j++) {
            w[i][j] = w[i - Nk][j] ^ temp[j];
        }
    }
    return w;
}
```

Рисунок 3.4 – Функція розширення ключа.

Також не менш важливими є функція зсуву рядків (рис. 3.5) та перемішування стовпців (рис. 3.6), які розроблені для пришвидшення і так вже швидкого алгоритму шифрування, без використання зайвих циклів і т.п.

```
function shiftRows(state) {
    let t = state[1];
    state[1] = state[5];
    state[5] = state[9];
    state[9] = state[13];
    state[13] = t;
    t = state[2];
    let t2 = state[6];
    state[2] = state[10];
    state[6] = state[14];
    state[10] = t;
    state[14] = t2;
    t = state[3];
    state[3] = state[15];
    state[15] = state[11];
    state[11] = state[7];
    state[7] = t;
}
```

Рисунок 3.5 – Функція перемішування рядків.

```
function mixColumns(state) {
    for (let c = 0; c < 4; c++) {
        const i = c * 4;
        const a0 = state[i], a1 = state[i + 1], a2 = state[i + 2], a3 = state[i + 3];
        const r0 = xtime(a0) ^ (a1 ^ xtime(a1)) ^ a2 ^ a3;
        const r1 = a0 ^ xtime(a1) ^ (a2 ^ xtime(a2)) ^ a3;
        const r2 = a0 ^ a1 ^ xtime(a2) ^ (a3 ^ xtime(a3));
        const r3 = (a0 ^ xtime(a0)) ^ a1 ^ a2 ^ xtime(a3);
        state[i] = r0;
        state[i + 1] = r1;
        state[i + 2] = r2;
        state[i + 3] = r3;
    }
}
```

Рисунок 3.6 – Функція перемішування стовпців.

Після шифрування пароля наступним важливим кроком у розробленому методі є використання ХФ для хешування зашифрованого тексту. Для цього було обрано алгоритм SHA-512, реалізація якого виконана за допомогою готових функцій бібліотеки crypto для Node.js (рис. 3.7).

```
function securePassword(password, ivProvided) {
    const crypted = aes.aesEncryptCBC(password, process.env.SECRET_KEY, ivProvided);
    const hashedPassword = crypto.createHash('sha512')
                                 .update(aes.bytesToHex(crypted.ciphertext))
                                 .digest('hex');
    return { hashedPassword, iv: crypted.iv };
}
```

Рисунок 3.7 – Функція шифрування та хешування паролів.

Як видно з рисунку 3.7 функція захисту паролів приймає сам пароль та якщо користувач вже існує, вектор ініціалізації, що йому було присвоєно під час реєстрації. Після отримання цих даних, пароль шифрується відповідно до того процесу шифрування, як вже було описано раніше у роботі. Потім отриманий шифротекст (послідовність байтів) конвертується в шістнадцяткове (HEX) представлення, і саме ця HEX-рядок хешується за допомогою SHA-512. В результаті виводиться значення 512-бітового хешу, який далі записується у базу даних.

Повний лістинг коду програмного модуля для шифрування паролів за допомогою AES можна переглянути у Додатку А.

3.3 Опис додаткових функціональних компонентів веб-додатку

Окрім основного рішення – розробленого методу захисту паролів – до веб-додатку було додано декілька інших функціональних можливостей, які покращують безпеку збереження паролів.

Однією з найважливіших таких функцій є обмеження кількості спроб введення даних облікового запису. Після певної кількості невдалих спроб входу доступ блокується (рис. 3.8).

```
const limiter = rateLimit({
    windowMs: 1 * 60 * 1000,
    max: 5,
    handler: (req, res, next, options) => {
        res.status(options.statusCode).json({ message: options.message });
    },
    message: 'Too many login attempts, please try again after 5 minutes',
    standardHeaders: true,
    legacyHeaders: false,
});
```

Рисунок 3.8 – Функція обмеження кількості спроб входу.

Додавання цього компоненту значно ускладнює використання атаки грубої сили (атаки перебору) для підбору паролів користувачів, оскільки після п'яти невдалих спроб входу доступ до облікового запису буде заблоковано. Тривалість блокування можна налаштовувати в діапазоні від 1 до 60 хвилин (наприклад, 1, 5, 15, 30 або 60 хвилин).

Іншим важливим доданим компонентом є перевірка на те, які символи вводяться у рядок імені користувача під час авторизації та реєстрації (рис. 3.9). Виконується це завдяки регулярним виразам, які якщо в імені присутні символи, окрім літер латинського алфавіту та чисел, а також довжина менша 6 символів, то в такому випадку видається помилка з вказівкою на це.

```
const regexUsername = /^[A-Za-z0-9_]+$/;
const lengthRegex = /^.{6,32}$/;
if (!regexUsername.test(username)) {
    const invalidChars = username.match(/[^A-Za-z0-9_]/g);
    return res.status(400).json({
        message: `Username contains forbidden character(s): ${invalidChars ? invalidChars.join(', ') : 'unknown'}`
    });
}
if (!lengthRegex.test(username)) {
    return res.status(400).json({ message: 'Username must be between 6 and 32 characters long'});
}
```

Рисунок 3.9 – Перевірка на правильність вводу імені користувача.

Ще одним важливим компонентом є перевірка чи підходить введений пароль користувачем під час реєстрації певним правилам (рис. 3.10).Такими правилами є: довжина пароля повинна бути не менше 8 символів, та використовуватися можуть лише літери латинського алфавіту разом із загальноприйнятими спеціальними символами (наприклад коми, точки, окличні знаки та знаки питання, символ “^” та інші).

```
const passwordRegex = /^[\x21-\x7E]{8,32}$/;
if (!passwordRegex.test(password)) {
    return res.status(400).json({
      message: 'Password must be at least 8 characters long and contain only letters, digits, punctuation and symbols'
    });
}
```

Рисунок 3.10 – Перевірка на правильність введення пароля.

Також варто описати наступний компонент – керування сесіями користувачів, який реалізований за допомогою бібліотеки express-session (рис. 3.11). За допомогою цього компоненту підтримується стан автентифікації користувача між різними запитами до сервера. При успішному вході до системи, ідентифікаційні дані користувача зберігаються у серверній сесії, яка асоціюється з унікальним ідентифікатором сесії, що передається клієнту у вигляді файлів cookie. Цілісність даних сесії забезпечується секретним ключем, який потрібен для підпису ідентифікатора сесії, тим самим запобігаючи підробці сесійних даних з боку зловмисника, ідентифікуючи користувача при наступних запитах.

```
app.use(session({
    secret: process.env.SESSION_SECRET || 'secret_key',
    resave: false,
    saveUninitialized: true
}));
```

Рисунок 3.11 – Функція керування сесіями користувачів.

Для того, щоб контролювати доступ до захищених ресурсів веб-додатку, було додано перевірку автентифікації (рис. 3.12). Основною задачею цього компонента є перевірка наявності активної та валідної сесії користувача, який намагається отримати доступ до певного ресурсу (маршруту). У розробленому

веб-додатку маршрутами були сторінки профіля користувача, або взагалі сторінка адміністратора.

Ця функція аналізує об'єкт сесії і якщо сесійні дані користувача існують, це свідчить про те, що користувач автентифікований, і обробка запиту передається далі за ланцюжком обробників. Якщо даних сесії про користувача не було знайдено, то функція повертає відповідь зі статусом 'Unauthorized' (відсутність доступу до ресурсу), тим самим блокуючи доступ неавторизованим особам.

```
function isAuthenticated(req, res, next) {
    if (req.session && req.session.user) {
        next();
    } else {
        res.status(401).json({ message: 'Unauthorized' });
    }
}
```

Рисунок 3.12 – Функція перевірки автентифікації.

Всі ці компоненти були реалізовані на серверній стороні веб-додатку, тому повний лістинг коду сервера наведений у Додатку Б.

3.4 Висновки до третього розділу

У даному розділі були описані головні засоби, які використовувалися для реалізації розробленого методу та разом з ним веб-застосунку, де він використовується. Такими засобами були: JavaScript (основна мова програмування), Node.js (середа виконання для JavaScript, який потрібен для створення серверної частини), Express.js (фреймворк для зручного управління та масштабування серверної частини), Mysql (база даних, де зберігаються дані захищені розробленим методом).

Було описана реалізація розробленого методу, тобто яким чином відбувається спочатку шифрування паролів, а потім їх хешування. Також були продемонстровані додаткові функціональні компоненти, які також можуть підвищувати безпеку веб-застосунку.

РОЗДІЛ 4
ДЕМОНСТРАЦІЯ МОЖЛИВОСТЕЙ РОЗРОБЛЕНОГО РІШЕННЯ

4.1 Порівняння існуючих рішень з розробленим методом

Для оцінки розробленого комбінованого методу AES-256 + SHA-512 було проведено його порівняння зі стандартними підходами SHA-512, bcrypt за такими показниками: навантаження на систему під час виконання хешування або шифрування, загальний час виконання та рівень захищеності. Окремо було протестовано швидкість чистого шифрування AES-256, щоб оцінити внесок цього етапу в загальну продуктивність комбінованого методу.

Для виконання перевірки були обчислені хеш-значення та виконано шифрування AES-256 для наступних наборів даних:

Набір 1 – 1 з розповсюджених паролів.

Набір 2 – 1 пароль, згенерований випадковим чином з використанням великих та малих літер.

Набір 3 – 500 поширених паролів зі словників.

Набір 4 – 500 паролів, згенерованих випадковим чином з використанням великих та малих літер.

Набір 5 – 15000 розповсюджених паролів.

Набір 6 – 20 згенерованих паролів довжиною 12 символів та з використанням великих/малих літер, цифр та спеціальних символів.

Процес хешування/шифрування та вимірювання навантажень відбувалися на комп’ютерній системі з такими характеристиками: центральний процесор Intel Core I7-10700KF, графічний процесор Nvidia GeForce GTX 1070 та операційна система Windows 11. Для перевірки стійкості до злому використовувався програмний засіб Hashcat версії 6.2.6. Версія мови програмування, на якій розроблено веб-застосунок та відбувалися всі заміри – Node.js 22.13.1.

4.1.1 Навантаження на систему та час виконання

Для всіх наборів були проведені заміри використання центрального процесора та оперативної пам'яті на початку процесу хешування/шифрування, максимальне та середнє значення під час роботи.

Для хешування/шифрування наборів з одного пароля навантаження на систему було не значним та зайняло не багато часу. Всі методи використовували менше 1% потужності системи (табл. 4.1) та виконувалися протягом від 1 мс до 200 мс (рис. 4.1).

Таблиця 4.1

Навантаження на процесор під час обробки 1 пароля.

	Початкове навантаження	**Середнє навантаження**	**Пікове навантаження**	**Кінцеве навантаження**
SHA-512	0.1 %	0.1 %	0.2 %	0.1 %
AES-256	0.2 %	0.3 %	0.4 %	0.3 %
AES-256+ SHA-512	0.1 %	0.4 %	0.5 %	0.4 %
Bcrypt (8 раундів)	0.1 %	0.7 %	1 %	0.5 %
Bcrypt (10 раундів)	0.2 %	0.9 %	1.1 %	0.4 %
Bcrypt (12 раундів)	0.1 %	1.1 %	1.3 %	0.4 %

Як видно з таблиці вище, bcrypt навіть при хешуванні одного пароля використовує більше ресурсів системи, аніж SHA-512, або AES-256+ SHA-512. Це пояснюється тим, що під час хешування за допомогою bcrypt значно більша кількість обчислювальних операцій, особливо при використанні 12 раундів.

Загальний час виконання також дуже різниться між цими функціями. AES-256 та SHA-512 за своєю природою є дуже швидкими, на відміну від них, час виконання bcrypt зростає експоненційно (приблизно вдвічі) зі збільшенням кількості раундів на одиницю. Тому на рисунку 4.1 помітно, наскільки довше виконується хешування одного пароля за допомогою bcrypt з 12 раундами (рекомендована кількість) порівняно з іншими методами, такими як bcrypt з меншою кількістю раундів або AES-256/SHA-512.

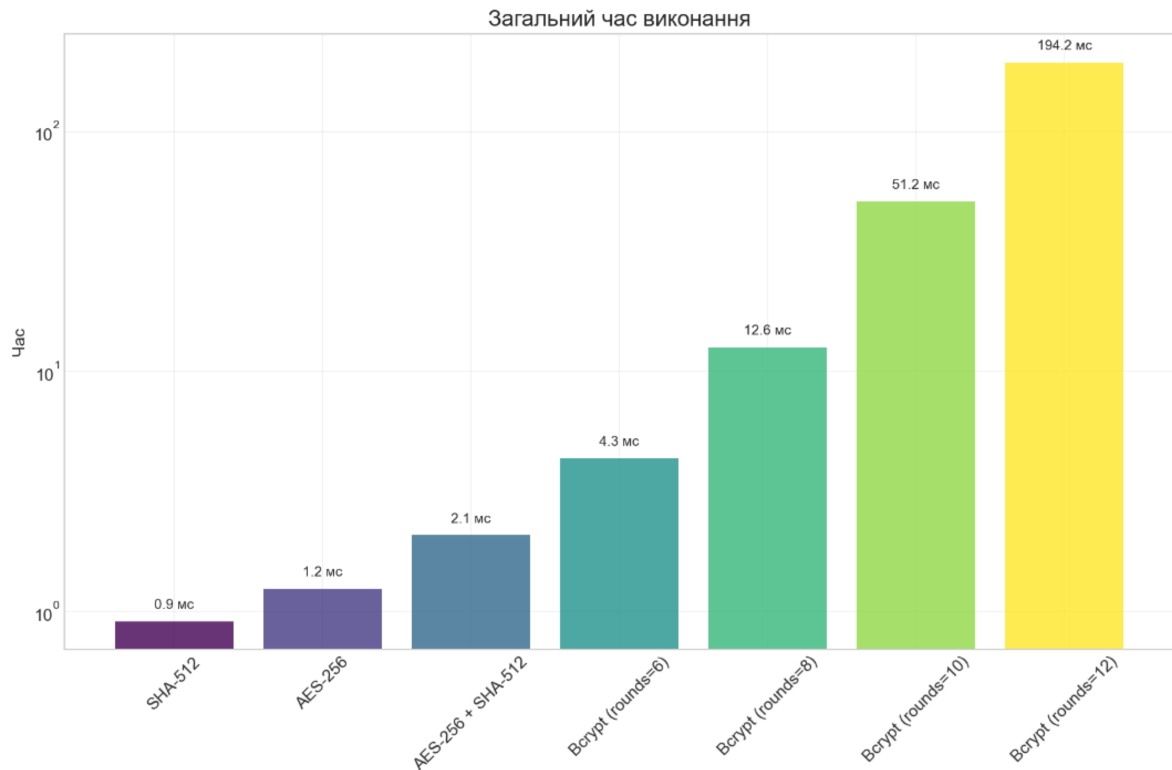


Рисунок 4.1 – Загальний час хешування/шифрування 1 пароля.

На відміну від ХФ Argon2, обрані методи не є залежними від використання великої кількості пам’яті, тому воно є незначним: найбільше значення було отримане при хешуванні 15000 паролів за допомогою bcrypt з 12 раундами, але навіть у цьому випадку воно не перевищувало 50 Мб.

При виконанні хешування/шифрування наборів з 500 паролів, отримані результати схожі з попередніми: bcrypt використовує значно більше ресурсів системи, аніж SHA-512, AES-256 чи комбінований метод AES-256 + SHA-512. Результати замірів використання ресурсів центрального процесора наведено у таблиці 4.2.

З результатів видно, що знову bcrypt використовує у багато разів більше системних ресурсів, аніж інші методи. Алгоритми SHA-512, AES-256 та їх поєднання є набагато менш вимогливими до ресурсів процесора, ніж bcrypt. Це зумовлено значно більшою кількістю обчислень, необхідних для хешування за допомогою bcrypt.

Таблиця 4.2

Навантаження на процесор під час обробки 500 паролів.

	Початкове навантаження	**Середнє навантаження**	**Пікове навантаження**	**Кінцеве навантаження**
SHA-512	0.1 %	0.6 %	0.8 %	0.1 %
AES-256	0.2 %	1.5 %	2 %	0.5 %
AES-256+ SHA-512	0.4 %	1.4 %	2.4 %	0.3 %
Bcrypt (8 раундів)	0.2 %	6.7 %	9.1 %	0.6 %
Bcrypt (10 раундів)	0.2 %	15.1 %	23.4 %	0.4 %
Bcrypt (12 раундів)	0.3 %	16.3 %	25 %	0.5 %

Результати вимірювання часу виконання цих процесів для 500 паролів також демонструють аналогічну тенденцію: bcrypt виконується набагато довше (рис. 4.2).

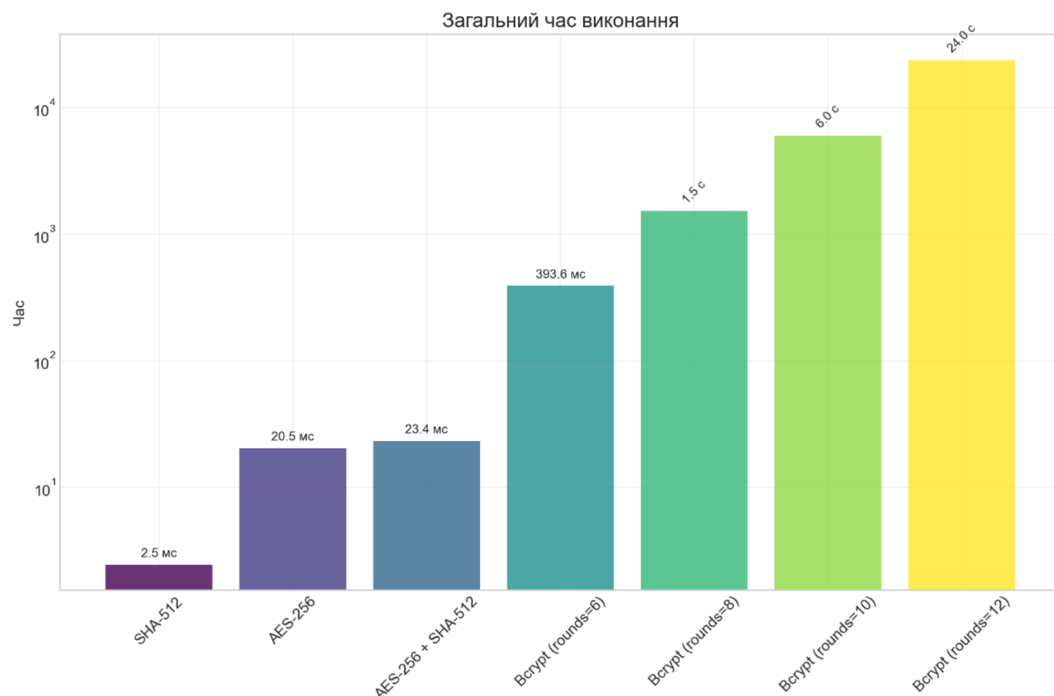


Рисунок 4.2 – Загальний час хешування/шифрування 500 паролів.

Результати вимірів використання процесора для хешування/шифрування 15000 паролів наведено у таблиці 4.3, а загальний час витрачений на ці операції зображено на рисунку 4.3.

Таблиця 4.3

Навантаження на процесор під час обробки 15000 паролів.

	Початкове навантаження	**Середнє навантаження**	**Пікове навантаження**	**Кінцеве навантаження**
SHA-512	0.1 %	0.4 %	0.5 %	0.1 %
AES-256	0.4 %	1 %	2.1 %	0.3 %
AES-256+ SHA-512	0.4 %	1.2 %	2.4 %	0.3 %
Bcrypt (8 раундів)	0.3 %	14.1 %	20.8 %	0.5 %
Bcrypt (10 раундів)	0.4 %	15 %	21.4 %	0.4 %
Bcrypt (12 раундів)	0.3 %	13 %	20.4 %	0.5 %

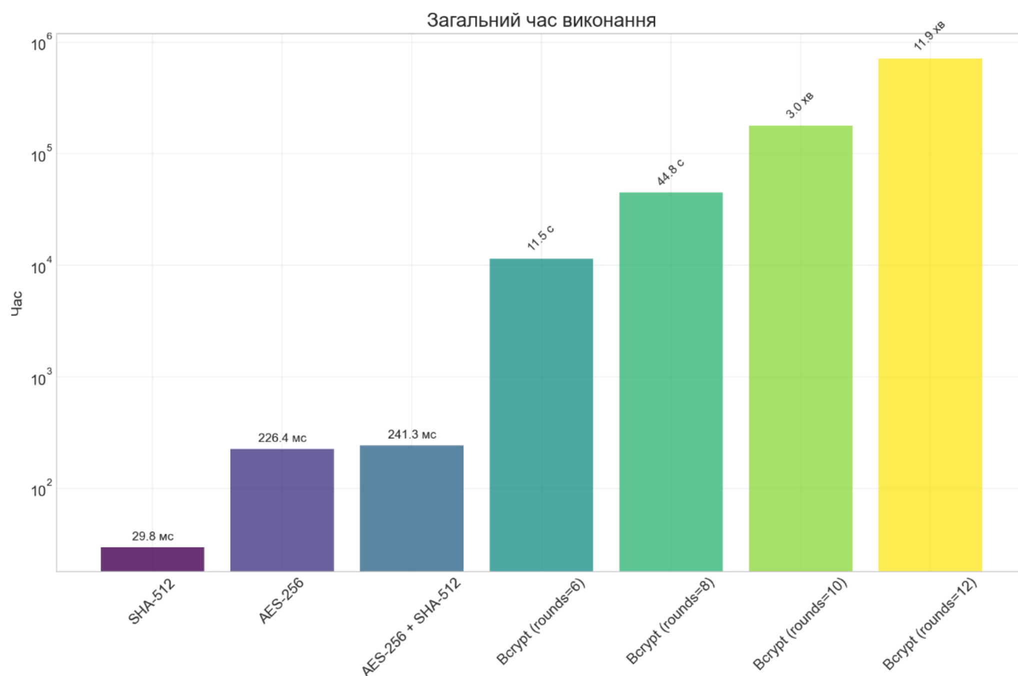


Рисунок 4.3 – Загальний час хешування/шифрування 15000 паролів.

Із отриманих результатів можна зробити висновки, що використання процесора при збільшенні кількості паролів, збільшується, але не надто сильно, та після певної кількості взагалі залишається майже сталою.

Стосовно часу, то чим більша кількість паролів для хешування, тим більше часу потрібно. Особливо це дуже помітно на прикладі bcrypt, де при збільшенні кількості паролів у 30 разів, час, необхідний для цього, також збільшився у 30

разів. У випадку SHA-512, AES-256 і комбінованого методу SHA-512+AES-256, при збільшенні кількості в 30 разів, необхідний час збільшився лише у 10-11 разів. У випадку збільшення паролів з 1 до 500, використаний час зріс не у 500 разів, а у 2.7 рази для SHA-512, та у 120 разів для bcrypt з 12 раундами.

4.1.2 Порівняння захищеності методів

Для аналізу захищеності розробленого методу було проведено атаки на набори раніше згенерованих хешів паролів. Для атаки були використані такі підходи: метод перебору за словником та метод повного перебору. Словник, що використовувався – Rockyou; він містить понад 14 мільйонів унікальних паролів, зібраних з близько 32 мільйонів облікових записів однойменної компанії.

Спершу було протестовано набори методом перебору за словником. Для знаходження одного розповсюдженого пароля у вигляді хешу SHA-512 було витрачено менше однієї секунди, його було знайдено майже моментально. Результати для bcrypt дещо відрізнялися: пароль, захешований за допомогою bcrypt (8 раундів), був знайдений також менш ніж за одну секунду, тоді як для 10 та 12 раундів було витрачено по 3 секунди на знаходження пароля (рис. 4.4).

```
Dictionary cache hit:
* Filename..: E:\Diploma\DiplomaCode\tests\passwords\rockyou.txt
* Passwords.: 14344384
* Bytes.....: 139921497
* Keyspace..: 14344384


Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: $2b$12$mr/vH90kFxIHFEYUz.0unetPsOiLMV1erPUSyoD3Qj7V...iWgZ0S
Time.Started.....: Tue Apr 22 15:45:21 2025, (3 secs)
Time.Estimated...: Tue Apr 22 15:45:24 2025, (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (E:\Diploma\DiplomaCode\tests\passwords\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      126 H/s (10.20ms) @ Accel:2 Loops:16 Thr:11 Vec:1
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 330/14344384 (0.00%)
Rejected.........: 0/330 (0.00%)
Restore.Point....: 0/14344384 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:4080-4096
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 -> cassie
Hardware.Mon.#1..: Temp: 53c Fan: 22% Util:100% Core:1911MHz Mem:3802MHz Bus:16

Started: Tue Apr 22 15:45:14 2025
Stopped: Tue Apr 22 15:45:25 2025
```

Рисунок 4.4 – Результат знаходження пароля для 12-раундового bcrypt.

При проведенні аналогічної операції для розробленого методу результат був очікуваним – пароль не було знайдено. Це пояснюється тим, що програма Hashcat обчислює хеші для паролів зі словника у їх відкритому вигляді, а не для результатів попереднього їх шифрування. На рисунку 4.5 можна побачити, що Hashcat виконав перевірку всіх 14 мільйонів паролів зі словника, але так і не знайшов відповідника для цільового хешу.

```
Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 1700 (SHA2-512)
Hash.Target......: 972d79033488784030bebcea14aa219ea169d2e22c467fc1e1e...74e461
Time.Started.....: Tue Apr 22 18:51:53 2025, (1 sec)
Time.Estimated...: Tue Apr 22 18:51:54 2025, (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (E:\Diploma\DiplomaCode\tests\passwords\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 19531.8 kH/s (5.37ms) @ Accel:1024 Loops:1 Thr:64 Vec:1
Recovered........: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.........: 14344384/14344384 (100.00%)
Rejected.........: 0/14344384 (0.00%)
Restore.Point....: 14344384/14344384 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: $HEX[30383434313332373933] -> $HEX[042a0337c2a156616d6f732103]
Hardware.Mon.#1..: Temp: 49c Fan:  0% Util: 26% Core:1733MHz Mem:3802MHz Bus:16

Started: Tue Apr 22 18:51:51 2025
Stopped: Tue Apr 22 18:51:54 2025
```

Рисунок 4.5 – Результат роботи програми для розробленого методу.

Для набору з 500 поширених паролів результати між методами вже помітно відрізняються. Наприклад, паролі, захешовані за допомогою SHA-512, були знайдені за 3 секунди, тоді як для хешів bcrypt (8 раундів) потрібно було вже майже 2 хвилини (рис. 4.6).

```
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: E:\Diploma\DiplomaCode\tests\encrypted_files\hashed_passwords_8_500_common.txt
Time.Started.....: Tue Apr 22 17:06:28 2025, (1 min, 52 secs)
Time.Estimated...: Tue Apr 22 17:08:20 2025, (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (E:\Diploma\DiplomaCode\tests\passwords\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     1968 H/s (10.23ms) @ Accel:2 Loops:16 Thr:11 Vec:1
Recovered........: 500/500 (100.00%) Digests (total), 500/500 (100.00%) Digests (new), 500/500 (100.00%) Salts
Progress.........: 328020/7172192000 (0.00%)
Rejected.........: 0/328020 (0.00%)
Restore.Point....: 330/14344384 (0.00%)
Restore.Sub.#1...: Salt:493 Amplifier:0-1 Iteration:240-256
Candidate.Engine.: Device Generator
Candidates.#1....: benfica -> cheyenne
Hardware.Mon.#1..: Temp: 58c Fan: 33% Util: 99% Core:1911MHz Mem:3802MHz Bus:16

Started: Tue Apr 22 17:06:20 2025
Stopped: Tue Apr 22 17:08:22 2025
```

Рисунок 4.6 – результат роботи програми для 500 розповсюджених паролів у 8-раундовому bcrypt.

Для bcrypt з 10 та 12 раундами витрачено набагато більше часу – 22 та 30 хвилин відповідно (рис. 4.7). За 6 хвилин атаки за словником було знайдено приблизно 100-120 паролів або ж 20-25% обома методами.

```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: E:\Diploma\DiplomaCode\tests\encrypted_files\hashed_passwords_12_500_common.txt
Time.Started.....: Tue Apr 22 17:36:30 2025, (6 mins, 54 secs)
Time.Estimated...: Thu Oct 08 01:16:06 2026, (1 year, 168 days)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (E:\Diploma\DiplomaCode\tests\passwords\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      125 H/s (10.28ms) @ Accel:2 Loops:16 Thr:11 Vec:1
Recovered........: 100/500 (20.00%) Digests (total), 100/500 (20.00%) Digests (new), 100/500 (20.00%) Salts
Progress.........: 51480/7172192000 (0.00%)
Rejected.........: 0/51480 (0.00%)
Restore.Point....: 0/14344384 (0.00%)
Restore.Sub.#1...: Salt:156 Amplifier:0-1 Iteration:624-640
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 -> cassie
Hardware.Mon.#1..: Temp: 59c Fan: 40% Util: 99% Core:1911MHz Mem:3802MHz Bus:16
```

Рисунок 4.7 – Знайдені 100 розповсюджених паролів у хеші bcrypt з 12 раундами.

Для розробленого методу ситуація аналогічна до минулого тесту – паролі не були знайдені, навіть після повної перевірки всього словника.

Під час атаки на набір з 15000 паролів зі словників, хеші SHA-512 для них були підібрані за 1 хвилину 20 секунд. Для хешів 8-раундового bcrypt за 6 хвилин було знайдено лише 45 паролів (~0.3% від усієї кількості). Для 12-раундового bcrypt за 10 хвилин було підібрано лише 7 паролів з 15000. Hashcat вміє вираховувати приблизний час для знаходження паролів. Для bcrypt з 12 раундами з набору такий час дорівнює більше 50 років (рис. 4.8).

```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: E:\Diploma\DiplomaCode\tests\encrypted_files\Bcrypt_common\hashed_passwords_12_15000_common.txt
Time.Started.....: Tue Apr 22 19:03:22 2025, (9 mins, 51 secs)
Time.Estimated...: Thu Aug 17 04:23:23 2079, (54 years, 116 days)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (E:\Diploma\DiplomaCode\tests\passwords\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      125 H/s (81.79ms) @ Accel:2 Loops:128 Thr:11 Vec:1
Recovered........: 7/15000 (0.05%) Digests (total), 7/15000 (0.05%) Digests (new), 7/15000 (0.05%) Salts
Remaining........: 14993 (99.95%) Digests, 14993 (99.95%) Salts
Recovered/Time...: CUR:1,N/A,N/A AVG:0.71,N/A,N/A (Min,Hour,Day)
Progress.........: 73920/215165760000 (0.00%)
Rejected.........: 0/73920 (0.00%)
Restore.Point....: 0/14344384 (0.00%)
Restore.Sub.#1...: Salt:224 Amplifier:0-1 Iteration:2048-2176
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 -> cassie
Hardware.Mon.#1..: Temp: 59c Fan: 40% Util:100% Core:1911MHz Mem:3802MHz Bus:16
```

Рисунок 4.8 – Пошук 15000 паролів у вигляді хешу 12-раундового Bcrypt.

Після перебору паролів за словником було виконано атаку повного перебору (брутфорс). Для цього використовувалися ті самі набори з

розповсюджених паролів (могли містити великі та малі літери, а також числа) та набори випадково згенерованих паролів, з використанням великих та малих літер.

Під час атаки повного перебору на хеші SHA-512 (набір з 500 поширених паролів) за приблизно 20 хвилин було знайдено 362 паролі довжиною від 4 до 7 символів, що становить 72% від загальної кількості (рис. 4.9). Для цього підбору було встановлено наступну маску: символи, що включають латинські літери обох регістрів та цифри, довжиною від 4 до 16 символів.

```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 1700 (SHA2-512)
Hash.Target......: E:\Diploma\DiplomaCode\tests\encrypted_files\SHA-512\hashed_passwords_sha512_500_common.txt
Time.Started.....: Wed Apr 23 15:33:25 2025 (12 mins, 21 secs)
Time.Estimated...: Wed Apr 23 17:02:56 2025 (1 hour, 17 mins)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?1?1?1?1?1?1?1 [7]
Guess.Charset....: -1 ?l?u?d, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue......: 4/13 (30.77%)
Speed.#1.........:   655.7 MH/s (11.72ms) @ Accel:8 Loops:256 Thr:256 Vec:1
Recovered........: 362/500 (72.40%) Digests (total), 362/500 (72.40%) Digests (new)
Progress.........: 485471846400/3521614606208 (13.79%)
Rejected.........: 0/485471846400 (0.00%)
Restore.Point....: 2027520/14776336 (13.72%)
Restore.Sub.#1...: Salt:0 Amplifier:146944-147456 Iteration:0-512
Candidate.Engine.: Device Generator
Candidates.#1....: bfEJ8w0 -> iiCT3RY
Hardware.Mon.#1..: Temp: 82c Fan: 98% Util: 99% Core:1847MHz Mem:3802MHz Bus:16
```

Рисунок 4.9 – 500 розповсюджених паролів методом повного перебору SHA-512.

За наступні 20 хвилин було знайдено ще 8 паролів. Таким чином, за 40 хвилин загалом було підібрано 370 паролів, або 74% від набору (рис. 4.10). Подальшу атаку було припинено, оскільки, за оцінками Hashcat, лише для завершення перебору паролів довжиною 7 символів знадобилося б понад годину, а для паролів довжиною 8 символів – значно більше часу.

```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 1700 (SHA2-512)
Hash.Target......: E:\Diploma\DiplomaCode\tests\encrypted_files\SHA-512\hashed_passwords_sha512_500_common.txt
Time.Started.....: Wed Apr 23 15:33:25 2025 (29 mins, 53 secs)
Time.Estimated...: Wed Apr 23 17:02:50 2025 (59 mins, 32 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?1?1?1?1?1?1?1 [7]
Guess.Charset....: -1 ?l?u?d, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue......: 4/13 (30.77%)
Speed.#1.........:   655.5 MH/s (11.77ms) @ Accel:8 Loops:256 Thr:256 Vec:1
Recovered........: 370/500 (74.00%) Digests (total), 370/500 (74.00%) Digests (new)
Progress.........: 1179710668800/3521614606208 (33.50%)
Rejected.........: 0/1179710668800 (0.00%)
Restore.Point....: 4945920/14776336 (33.47%)
Restore.Sub.#1...: Salt:0 Amplifier:62464-62976 Iteration:0-512
Candidate.Engine.: Device Generator
Candidates.#1....: 7bhZFyu -> x3dLbJL
Hardware.Mon.#1..: Temp: 82c Fan: 98% Util: 99% Core:1835MHz Mem:3802MHz Bus:16
```

Рисунок 4.10 – Результат роботи за 40 хвилин атаки повним перебором SHA-512.

Під час атаки на хеші bcrypt (10 раундів) за 30 хвилин не було знайдено жодного пароля, а процес перебору все ще залишався на етапі перевірки паролів довжиною 4 символи (рис. 4.11). Це пов’язано з тим, що bcrypt є вимогливим до пам'яті, зокрема при спробах прискорення атаки за допомогою графічних процесорів. Для ефективного обчислення хешу bcrypt (який використовує ітерації шифру Blowfish) потрібно понад 4 кілобайти швидкої пам'яті на кожну паралельну обробку. Сучасні графічні процесори не мають такої кількості швидкої пам'яті на одне обчислювальне ядро, що обмежує ступінь паралелізму та ефективність атаки на GPU. Тому ця властивість bcrypt забезпечує певний захист від масового паралельного перебору на GPU, однак цей параметр є фіксованим, і майбутнє апаратне забезпечення може подолати це обмеження [47].

```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: E:\Diploma\DiplomaCode\tests\encrypted_files\Bcrypt_common\hashed_passwords_10_500_common.txt
Time.Started.....: Wed Apr 23 16:05:39 2025 (29 mins, 4 secs)
Time.Estimated...: Mon Oct 13 05:17:49 2025 (172 days, 12 hours)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?1?1?1?1 [4]
Guess.Charset....: -1 ?l?u?d, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue......: 1/13 (7.69%)
Speed.#1.........:      496 H/s (10.36ms) @ Accel:2 Loops:16 Thr:11 Vec:1
Recovered........: 0/500 (0.00%) Digests (total), 0/500 (0.00%) Digests (new), 0/500 (0.00%) Salts
Progress.........: 863610/7388168000 (0.01%)
Rejected.........: 0/863610 (0.00%)
Restore.Point....: 0/238328 (0.00%)
Restore.Sub.#1...: Salt:42 Amplifier:13-14 Iteration:864-880
Candidate.Engine.: Device Generator
Candidates.#1....: kari -> ksdr
Hardware.Mon.#1..: Temp: 59c Fan: 40% Util:100% Core:1898MHz Mem:3802MHz Bus:16
```

Рисунок 4.11 – Використання методу повного перебору для 10-раундового bcrypt.

При тестуванні розробленого методу за 30 хвилин не було знайдено жодного пароля, які були ним захешовані (рис. 4.12).

```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 1700 (SHA2-512)
Hash.Target......: E:\Diploma\DiplomaCode\tests\encrypted_files\AES-256+SHA-512\aes_sha512_passwords_500_common.txt
Time.Started.....: Wed Apr 30 16:31:04 2025, (38 mins, 14 secs)
Time.Estimated...: Wed Apr 30 18:00:49 2025, (51 mins, 31 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?1?1?1?1?1?1?1 [7]
Guess.Charset....: -1 ?l?u?d, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue......: 4/13 (30.77%)
Speed.#1.........:   653.8 MH/s (11.94ms) @ Accel:4 Loops:512 Thr:256 Vec:1
Recovered........: 0/500 (0.00%) Digests (total), 0/500 (0.00%) Digests (new)
Progress.........: 1500540764160/3521614606208 (42.61%)
Rejected.........: 0/1500540764160 (0.00%)
Restore.Point....: 6266880/14776336 (42.41%)
Restore.Sub.#1...: Salt:0 Amplifier:113408-113536 Iteration:0-128
Candidate.Engine.: Device Generator
Candidates.#1....: d5wqrgc -> kz3T2zn
Hardware.Mon.#1..: Temp: 82c Fan: 85% Util: 99% Core:1822MHz Mem:3802MHz Bus:16
```

Рисунок 4.12 – Повний перебір паролів для розробленого методу.

При тестуванні SHA-512 для випадково згенерованих паролів з використанням лише великих та малих літер і довжиною від 7 до 15 символів за 25 хвилин було знайдено 48 паролів з 500, що становить приблизно 10%.

Для хешів ХФ bcrypt (10 раундів) від аналогічного набору випадкових паролів за ті ж 25 хвилин не було знайдено жодного пароля, а орієнтовний час перебору лише 7-символьних паролів (за оцінкою Hashcat) перевищував 10 років.

Стосовно останнього набору з 20 паролів з великою ентропією (великі та малі літери, числа, спеціальні символи та з довжиною в 12 символів), Hashcat видав помилку про переповнення цілого числа. Це сталося, оскільки загальна кількість можливих комбінацій паролів (розмір простору ключів) для такої довжини та набору символів (89^{12}) перевищує максимальне значення, яке може зберігати стандартний 64-бітний беззнаковий цілочисельний тип даних, що використовується Hashcat для представлення розміру простору ключів.

Отже, провівши аналіз всіх вище описаних тестувань, можна побачити, що при атаці за словником звичайний SHA-512 є дуже вразливим алгоритмом, бо його обчислення відбувається швидко, навіть на не найкращому графічному процесорі. bcrypt при такій самій атаці показує себе краще, бо для цього потрібно зловмиснику витратити більше часу для знаходження паролів. Але навіть при збільшенні часу на пошук все одно є можливість розкриття поширених та слабких паролів. Використовуючи розроблений метод жодне тестування не мало можливості знайти правильні паролі. Як згадувалося раніше, це відбувається тому що, Hashcat та всі подібні йому програми виконують хешування для умовно «відкритих» паролів, тобто для певного набору символів довжиною в середньому від 10 до 14 символів. У випадку розробленого методу, хешування відбувається над рядком зашифрованого тексту (довжина рядка може бути від 32 до 64 символів в залежності чи зберігається IV чи ні), що і унеможливлює знаходження пароля зловмисником без знання того, що окрім звичайного хешування існує інший шар захисту у вигляді шифрування.

Також варто згадати, що bcrypt може бути вразливим до атак з використанням спеціалізованого апаратного забезпечення, такого як FPGA та

ASIC. Існує інструмент аудиту безпеки та відновлення паролів John The Ripper, який підтримує роботу з платами FPGA.

Станом на 2011 рік такі плати дозволяли ефективно запускати, наприклад, 124 оптимізовані ядра для обчислення bcrypt. Продуктивність таких FPGA-рішень саме для bcrypt могла перевищувати показники сучасних графічних процесорів, таких як Nvidia RTX 2080 Ti, через особливості алгоритму bcrypt, вимогливого до пам'яті. Порівняння продуктивності плат FPGA з Nvidia RTX 2080 Ti та центральним процесором AMD EPYC 7401P наведено в таблиці 4.4 [48].

Таблиця 4.4

Порівняння продуктивності FPGA з центральним та графічним процесорами.

Вартість	**ЦП Хешів/с**	**ГП Хешів/с**	**FPGA Хешів/с**
5	25200	28000	~120000
8	3150	3500	~15000
10	788	875	~3700
12	200	220	~1000

Також у порівнянні з SHA-512 та розробленого комбінованого методу, bcrypt є набагато більш вимогливий до ресурсів системи під час процесу хешування, а також вимагає набагато більше часу для обробки даних. За таких умов його можуть не використовувати розробники більш малих з обмеженими ресурсами веб-застосунків.

4.2 Висновки четвертого до розділу

У даному розділі було проведено детальний аналіз та порівняння розробленого методу з іншими сучасними хеш-функціями – SHA-512 та bcrpyt. В результаті було виявлено, що SHA-512 є досить слабким методом захисту, якщо

використовувати або найбільш розповсюджені (зазвичай дуже легкі та які легко запам'ятати) або короткі паролі. Перший вид легко знаходиться за допомогою атак за словником або гібридних, інший дуже швидко підбриється методом грубого підбору. Також SHA-512 є дуже швидким алгоритмом та не потребує використання великої кількості системних ресурсів.

bcrypt запобігає такі види атак завдяки своїй вартості (кількості ітерацій, які потрібно виконати перш ніж буде отримане хеш-значення). Збільшуючи значення вартості тим самим збільшується час, який необхідний для підбору правильного значення. В порівнянні з SHA-512, bcrypt вимагає набагато більшого використання центрального процесора з оперативною пам'яттю, а також час витрачений на хешування є в сотні разів більший за час, витрачений розробленим методом або SHA-512. Також bcrypt є вразливим до атак з використанням плат FPGA або ASIC.

Аналіз розробленого методу зберігання показав, що він є швидким та не вимагає великої кількості системних ресурсів. Захист у нього досягається тим, що хешування відбувається не над самим паролем, а над результатом шифрування цього пароля, що унеможливлює будь-які атаки підбором.

ВИСНОВКИ

Результатом роботи є розроблений новий метод збереження паролів у базах даних веб-застосунків, який забезпечує стійкість до їх компрометації та імплементовано його. Також було виконано порівняння його з вже існуючими, в результаті чого було виявлено, що розроблений метод є більш стійким та універсальним. Не менш важливим було проаналізовано питання законодавчої та нормативної бази країн світу, зокрема України. У ході роботи було досліджено сучасні загрози та вразливості, властиві паролям та при їх збереженні у базах даних. Також було виконано аналіз інших методів збереження паролів (використання солі, розтягування ключів тощо). Було виконано огляд алгоритмів, які використовувалися для розробки методу.

У першій частині роботи було проведеного дослідження нормативної та законодавчої бази. В результаті було встановлено, що безпечне зберігання паролів є одним із ключових аспектів забезпечення захисту інформаційних систем та персональних даних користувачів. Зокрема було досліджено положення GDPR, стандарт ISO/IEC 27001 та відповідні закони України – ЗУ "Про захист персональних даних", ЗУ "Про захист інформації в інформаційно-комунікаційних системах", ЗУ "Про основні засади забезпечення кібербезпеки України".

Проведений аналіз типових загроз та вразливостей, пов’язаних з неналежним зберіганням та використанням паролів, підкреслив актуальність та важливість застосування надійних криптографічних або інших методів захисту та зберігання паролів.

У другій частині роботи було розглянуто та детально описані сучасні алгоритми хешування і шифрування паролів, такі як SHA-512, bcrypt та AES-256. Розглянуто їхні технічні особливості, рівень криптографічної стійкості до відомих методів атак, а також практичні аспекти їхнього використання в системах захисту інформації. Визначено, що кожен із цих алгоритмів має свої переваги та

недоліки, а вибір конкретного алгоритму залежить від специфічних вимог до безпеки та продуктивності системи.

В третій частині роботи було проведено опис засобів реалізації, а також продемонстровано процес розробленого методу – комбінування шифрування та хешування для покращення безпеки паролів збережених у базі даних. Додатково було описано інші функціональні компоненти, які можуть покращувати безпеку веб-застосунку загалом.

У четвертій частині було проведено порівняння розробленого методу з вже існуючими. Результатом порівняння є те, що SHA-512 не надійна хеш-функція, при використанні слабких паролів; bcrypt достатньо захищена від зламу, але потребує багато ресурсів та часу; розроблений метод (комбінація SHA-512 та AES-256) є захищеним, не потребує великої кількості системних ресурсів та швидким.

Враховуючи встановлену мету кваліфікаційної роботи, були виконані наступні завдання:

- Проаналізовано сучасні загрози та вразливості при використанні та збереженні паролів.
- Визначено сучасні методи захисту паролів у базах даних.
- Розроблено метод та імплементувано його у веб-застосунок.
- Виконано порівняння з іншими методами збереження чутливих даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Загальний регламент про захист даних (GDPR), стаття 5 : норм.-прав. акт Європ. Союзу від 16.04.2016.

2. Загальний регламент про захист даних (GDPR), стаття 25 : норм.-прав. акт Європ. Союзу від 16.04.2016.

3. Загальний регламент про захист даних (GDPR), стаття 32 : норм.-прав. акт Європ. Союзу від 16.04.2016.

4. What is Pseudonymization? Techniques and Best Practices. VAADATA - Ethical Hacking Services. URL: https://www.vaadata.com/blog/what-is-pseudonymisation-techniques-and-best-practices/ (дата звернення 12.05.2025).

5. ISO/IEC 27001:2022. Вид. офіц. Швейцарія, 2022.

6. Про захист персональних даних: Закон України від 01.06.2010 № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text (дата звернення 12.05.2025).

7. Про захист інформації в інформаційно-телекомунікаційних системах: Закон України від 05.07.1994 № 80/94-ВР. URL: https://zakon.rada.gov.ua/laws/show/80/94-вр#Text (дата звернення 12.05.2025).

8. Про основні засади забезпечення кібербезпеки України: Закон України від 05.10.2017 № 2163-VIII. URL: https://zakon.rada.gov.ua/laws/show/2163-19#Text (дата звернення 12.05.2025).

9. PCI Compliance: The 12 Requirements and Compliance Checklist. Exabeam. URL: https://www.exabeam.com/explainers/pci-compliance/pci-compliance-a-quick-guide/ (дата звернення 12.05.2025).

10. Chanda K. Password Security: An Analysis of Password Strengths and Vulnerabilities. International Journal of Computer Network and Information Security. 2016. Т. 8, № 7. С. 23–30.

11. Kanta A., Coisel I., Scanlon M. A comprehensive evaluation on the benefits of context based password cracking for digital forensics. Journal of Information Security and Applications. 2024. Т. 84. С. 103809.

12. How Secure Is My Password? | Password Strength Checker. Security.org. URL: https://www.security.org/how-secure-is-my-password/ (дата звернення 12.05.2025).

13. Круглікова А. Д. Аналіз алгоритмів для оцінки стійкості паролів. Новітні інформаційні системи та технології. 2016. № 5.

14. William E. Burr, Donna F. Dodson, Elaine M. Newton, Ray A. Perlner. Electronic Authentication Guideline. Gaithersburg : NIST, 2011. 123 с.

15. Security habits around the world: A closer look at password security statistics | Bitwarden. Bitwarden. URL: https://bitwarden.com/blog/a-closer-look-at-password-statistics/ (дата звернення 12.05.2025).

16. Harris Poll. Online Security Survey Google. URL: https://services.google.com/fh/files/blogs/google_security_infographic.pdf (дата звернення 12.05.2025).

17. What is Phishing? Types of Phishing Attacks | Fortinet. Fortinet. URL: https://www.fortinet.com/resources/cyberglossary/phishing (дата звернення 12.05.2025).

18. IBM. What is Phishing? | IBM. IBM - United States. URL: https://www.ibm.com/think/topics/phishing (дата звернення 12.05.2025).

19. Man in the Middle (MITM) Attack. CrowdStrike: We Stop Breaches with AI-native Cybersecurity. URL: https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/man-in-the-middle-mitm-attack/ (дата звернення 12.05.2025).

20. Authentication Vulnerabilities and Defense. IGNOU. eGyanKosh. URL: https://www.egyankosh.ac.in/bitstream/123456789/96531/1/Unit-1.pdf (дата звернення 12.05.2025).

21. Password Storage. OWASP Cheat Sheet Series. URL: https://owasp.deteact.com/cheat/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення 12.05.2025).

22. Digital identity guidelines: authentication and lifecycle management / P. A. Grassi та ін. Gaithersburg, MD : National Institute of Standards and Technology, 2017.

23. Що таке Key Stretching – Терміни та визначення в кібербезпеці. VPN Unlimited – Fast & Secure VPN service. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/key-stretching (дата звернення 12.05.2025).

24. Kaliski, B. (2000). RFC 2898: PKCS #5: Password-Based Cryptography Specification.

25. Provos N., Mazières D. A Future-Adaptable Password Scheme // Proceedings of the 1999 USENIX Annual Technical Conference. USENIX Association, 1999. С. 81–91.

26. Percival C. Stronger Key Derivation via Sequential Memory-Hard Functions. Технічний звіт, 2009.

27. Password Hashing Competition. Password Hashing Competition. URL: https://www.password-hashing.net/ (дата звернення 12.05.2025).

28. Biryukov A., Dinu D., Khovratovich D. Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications. 2016 IEEE European Symposium on Security and Privacy (EuroS&P), м. Saarbrucken, 21–24 берез. 2016 р.

29. What is symmetric encryption? ENTRUST. URL: https://www.entrust.com/resources/learn/symmetric-encryption (дата звернення 12.05.2025).

30. Brush K., Cobb M. What is Asymmetric Cryptography? Definition from SearchSecurity. Search Security. URL: https://www.techtarget.com/searchsecurity/definition/asymmetric-cryptography (дата звернення 12.05.2025).

31. What is the role of IV in stream ciphers? Why is it needed and what is its importance? Cryptography Stack Exchange. URL:

https://crypto.stackexchange.com/questions/55921/what-is-the-role-of-iv-in-stream-ciphers-why-is-it-needed-and-what-is-its-impor (дата звернення 12.05.2025).

32. Symmetric Key Cryptography - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/symmetric-key-cryptography/ (дата звернення 12.05.2025).

33. Hash Function. Corporate Finance Institute. URL: https://corporatefinanceinstitute.com/resources/cryptocurrency/hash-function/ (дата звернення 12.05.2025).

34. Janki Mehta. SHA1 vs SHA2 vs SHA256 vs SHA512 Hash Algorithms – Know the Difference. Certera. URL: https://certera.com/blog/sha1-vs-sha2-vs-sha256-vs-sha512-hash-algorithms-know-the-difference/ (дата звернення 12.05.2025).

35. Rhodes D. SHA-512 Hashing Algorithm Overview. Komodo Academy | En. URL: https://komodoplatform.com/en/academy/sha-512/#:~:text=SHA-512,%20or%20Secure%20Hash,hashing,%20and%20digital%20record%20verification (дата звернення 12.05.2025).

36. Eastlake D., Hansen T. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC Editor, 2011.

37. Hash Algorithm – Cryptography and Network. e-Adhyayan | Books for PG Courses. URL: https://ebooks.inflibnet.ac.in/csp11/chapter/hash-algorithm/ (дата звернення 12.05.2025).

38. Anand A. Breaking Down : SHA-512 Algorithm. Medium. URL: https://infosecwriteups.com/breaking-down-sha-512-algorithm-1fdb9cc9413a (дата звернення 12.05.2025).

39. Awati R., Bernstein C., Cobb M. What is the Advanced Encryption Standard (AES)? | Definition from TechTarget. Search Security. URL: https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard (дата звернення 12.05.2025).

40. Advanced Encryption Standard - Decryption, Algorithm - BitcoinWiki. BitcoinWiki. URL: https://bitcoinwiki.org/wiki/advanced-encryption-standard (дата звернення 12.05.2025).

41. Dworkin M. J. Advanced Encryption Standard (AES). Gaithersburg, MD : National Institute of Standards and Technology, 2023.

42. Dworkin M. J. Recommendation for block cipher modes of operation. Gaithersburg, MD : National Institute of Standards and Technology, 2010.

43. McPeak J., Wilton P. Beginning JavaScript. 5-те вид. Wiley & Sons, Incorporated, 2015. 768 с.

44. Wexler J. Get Programming with Node.js. Manning Publications Co. LLC, 2019. 463 с.

45. Hahn E. Express in Action: Writing, building, and testing Node.js applications. Manning Publications, 2016. 256 с.

46. DuBois P. MySQL. 5-те вид. Addison-Wesley, 2013. 1184 с.

47. Password hashing methods and algorithms on the .net platform / V. Fedorchenko та ін. Advanced information systems. 2024. Т. 8, № 4. С. 82–92.

48. ScatteredSecrets.com. Bcrypt password cracking extremely slow? Not if you are using hundreds of FPGAs! Medium. URL: https://scatteredsecrets.medium.com/bcrypt-password-cracking-extremely-slow-not-if-you-are-using-hundreds-of-fpgas-7ae42e3272f6 (дата звернення 12.05.2025).

49. Горбатюк П., Пархоменко І. Метод захисту паролів у базах даних веб-застосунків. VIII Міжнародна науково-практична конференція «Проблеми кібербезпеки інформаційно-комунікаційних систем» (PCSICS). 2025. С. 175-176.

ДОДАТОК А

Лістинг частини коду програмного модуля для шифрування паролів за допомогою AES-256.

```
function keyExpansion(key) {
    const Nk = 8;
    const Nb = 4;
    const Nr = 14;
    const w = [];
    let temp = [];

    for (let i = 0; i < Nk; i++) {
        w[i] = [key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i + 3]];
    }
    for (let i = Nk; i < Nb * (Nr + 1); i++) {
        temp = w[i - 1].slice();
        if (i % Nk === 0) {
            temp = subWord(rotWord(temp));
            temp[0] ^= Rcon[i / Nk];
        } else if (i % Nk === 4) {
            temp = subWord(temp);
        }
        w[i] = [];
        for (let j = 0; j < 4; j++) {
            w[i][j] = w[i - Nk][j] ^ temp[j];
        }
    }
    return w;
}
function getRoundKey(expandedKey, round) {
    const roundKey = [];
    for (let i = 0; i < 4; i++) {
```

```
            roundKey.push(...expandedKey[round * 4 + i]);
        }
        return roundKey;
    }
    function mixColumns(state) {
        for (let c = 0; c < 4; c++) {
            const i = c * 4;
            const a0 = state[i], a1 = state[i + 1], a2 = state[i + 2],
a3 = state[i + 3];
            const r0 = xtime(a0) ^ (a1 ^ xtime(a1)) ^ a2 ^ a3;
            const r1 = a0 ^ xtime(a1) ^ (a2 ^ xtime(a2)) ^ a3;
            const r2 = a0 ^ a1 ^ xtime(a2) ^ (a3 ^ xtime(a3));
            const r3 = (a0 ^ xtime(a0)) ^ a1 ^ a2 ^ xtime(a3);
            state[i] = r0;
            state[i + 1] = r1;
            state[i + 2] = r2;
            state[i + 3] = r3;
        }
    }
    function aesEncryptBlock(input, key) {
        let state = input.slice();
        const expandedKey = keyExpansion(key);
        const Nr = 14;
        addRoundKey(state, getRoundKey(expandedKey, 0));
        for (let round = 1; round < Nr; round++) {
            subBytes(state);
            shiftRows(state);
            mixColumns(state);
            addRoundKey(state, getRoundKey(expandedKey, round));
        }
        subBytes(state);
        shiftRows(state);
        addRoundKey(state, getRoundKey(expandedKey, Nr));
        return state;
    }
```

```
function padPKCS7(bytes) {
    const padding = 16 - (bytes.length % 16);
    for (let i = 0; i < padding; i++) {
        bytes.push(padding);
    }
    return bytes;
}
function unpadPKCS7(bytes) {
    const padding = bytes[bytes.length - 1];
    return bytes.slice(0, bytes.length - padding);
}
function stringToBytes(str) {
    const bytes = [];
    for (let i = 0; i < str.length; i++) {
        bytes.push(str.charCodeAt(i) & 0xFF);
    }
    return bytes;
}
function bytesToString(bytes) {
    return String.fromCharCode(...bytes);
}
function bytesToHex(bytes) {
    return bytes.map(b => ('0' + b.toString(16)).slice(-2)).join('');
}
function hexToBytes(hex) {
    const bytes = [];
    for (let c = 0; c < hex.length; c += 2) {
        bytes.push(parseInt(hex.substr(c, 2), 16));
    }
    return bytes;
}
function aesEncryptCBC(message, keyString, ivProvided) {
    const key = stringToBytes(keyString);
    let bytes = stringToBytes(message);
    padPKCS7(bytes);
```

```
    let iv;
    if (ivProvided) {
        iv = ivProvided;
    } else {
        const ivBuffer = crypto.randomBytes(16);
        iv = Array.from(ivBuffer);
    }
    let prev = iv.slice();
    let ciphertext = [];
    for (let i = 0; i < bytes.length; i += 16) {
        const block = bytes.slice(i, i + 16);
        const xored = xorBytes(block, prev);
        const encryptedBlock = aesEncryptBlock(xored, key);
        ciphertext = ciphertext.concat(encryptedBlock);
        prev = encryptedBlock;
    }
    return { iv, ciphertext };
}
```

ДОДАТОК Б

Частковий лістинг коду серверної частини веб-додатку

```
const express = require('express');
const session = require('express-session');
const rateLimit = require('express-rate-limit');
const crypto = require('crypto');
const path = require('path');
const dotenv = require('dotenv');
const db = require('./config/db');
const aes = require('./aes');

dotenv.config();
const app = express();

app.use(express.json());
app.use(express.urlencoded({ extended: false }));

app.use(session({
    secret: process.env.SESSION_SECRET || 'secret_key',
    resave: false,
    saveUninitialized: true
}));

app.use(express.static(path.join(__dirname, '../public')));
app.set('trust proxy', 1);

function isAuthenticated(req, res, next) {
    if (req.session && req.session.user) {
        next();
    } else {
        res.status(401).json({ message: 'Unauthorized' });
    }
```

```
    }

    function isAdmin(req, res, next) {
        if (req.session && req.session.user && req.session.user.role ===
'admin') {
            next();
        } else {
            res.status(403).json({ message: 'Forbidden: Admins only' });
        }
    }
    function securePassword(password, ivProvided) {
            const     crypted     =     aes.aesEncryptCBC(password,
process.env.SECRET_KEY, ivProvided);
        const hashedPassword = crypto.createHash('sha512')
                             .update(aes.bytesToHex(crypted.ciphertext))
                              .digest('hex');
        return { hashedPassword, iv: crypted.iv };
    }

    const limiter = rateLimit({
        windowMs: 1 * 60 * 1000,
        max: 5,
        handler: (req, res, next, options) => {
                res.status(options.statusCode).json({    message:
options.message });
        },
         message: 'Too many login attempts, please try again after 5
minutes',
        standardHeaders: true,
        legacyHeaders: false,
    });

    app.post('/register', async (req, res) => {
        const { username, email, password } = req.body;
        if (!username || !email || !password) {
```

```
            return res.status(400).json({ message: 'All fields required'
});
        }
        const regexUsername = /^[A-Za-z0-9_]+$/;
        const lengthRegex = /^.{6,32}$/;
        if (!regexUsername.test(username)) {
            const invalidChars = username.match(/[^A-Za-z0-9_]/g);
            return res.status(400).json({
                message: `Username contains forbidden character(s):
${invalidChars ? invalidChars.join(', ') : 'unknown'}`
            });
        }
        if (!lengthRegex.test(username)) {
            return res.status(400).json({ message: 'Username must be
between 6 and 32 characters long'});
        }
        const regexEmail = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!regexEmail.test(email)) {
            return res.status(400).json({ message: 'Invalid email
address' });
        }
        const passwordRegex = /^[\x21-\x7E]{8,32}$/;
        if (!passwordRegex.test(password)) {
            return res.status(400).json({
              message: 'Password must be at least 8 characters long and
contain only letters, digits, punctuation and symbols'
            });
        }
        db.query('SELECT * FROM users WHERE username = ? OR email = ?',
[username, email], async (err, results) => {
            if (err) return res.status(500).json({ message: 'Database
error' });
            if (results.length > 0) {
                return res.status(400).json({ message: 'User already
exists' });
```

```
            }
            const { hashedPassword, iv } = securePassword(password);
            db.query('INSERT INTO users (username, email, password, role) VALUES (?, ?, ?, ?)', [username, email, hashedPassword, 'user'], (err, result) => {
                if (err) return res.status(500).json({ message: 'Database error' });
                db.query('INSERT INTO user_iv (iv, user_id) VALUES (?, ?)', [aes.bytesToHex(iv), result.insertId], (err) => {
                    if (err) return res.status(500).json({ message: 'Database error' });
                    res.status(201).json({ message: 'User registered successfully' });
                });
                //res.status(201).json({ message: 'User registered successfully' });
            });
        });
    });
```