Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(освітній рівень)

на тему: "Дослідження сучасних алгоритмів хешування паролів користувачів"

Виконав: студент (ка) 4 курсу, групи СБ - 41
Спеціальності:
125 «Кібербезпека»
(шифр і назва напряму підготовки, спеціальності)
Шарик Олександр Володимирович
підпис (прізвище та ініціали)

Керівник Загородна Н.В.
підпис (прізвище та ініціали)
Нормоконтроль Тимощук Д. І.
підпис (прізвище та ініціали)

Завідувач кафедри Загородна Н.В.
підпис (прізвище та ініціали)

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____________ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _________ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека
(шифр і назва спеціальності)

Студенту Шарику Олександру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження сучасних алгоритмів хешування паролів користувачів

Керівник роботи Загородна Наталя Володимирівна, к.т.н., доцент, завідувач кафедри КБ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «15» 04 2024 року № 4/7-350

2. Термін подання студентом завершеної роботи 09.06.2024

3. Вихідні дані до роботи Характеристики алгоритмів хешування паролів користувачів

4. Зміст роботи (перелік питань, які потрібно розробити)
Знайти та описати алгоритми, що використовуються при хешуванні паролів користувачів.
Проаналізувати властивості, характеристику та принцип роботи алгоритмів хешування.
Розробити програму, що реалізує досліджуванні алгоритми хешування та виконує заміри їхньої ефективності.
Безпека життєдіяльності, основи охорони праці.
Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Мариненко С.Ю, к.т.н., доцент кафедри МТ		

7. Дата видачі завдання 29.01.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	29.01 – 02.02	Виконано
2.	Підбір літературних джерел	03.02 – 06.02	Виконано
3.	Опрацювання джерел в галузі дослідження	07.02 – 21.02	Виконано
4.	Вибір програмних засобів розробки	22.02 – 14.03	Виконано
5.	Розроблення програмного коду	15.03 – 24.03	Виконано
6.	Тестування роботи програми	25.03 – 08.04	Виконано
7.	Оформлення розділу «Аналіз проблеми збереження паролів користувачі»	10.04 – 25.04	Виконано
8.	Оформлення розділу «Аналіз сучасних алгоритмів хешування для зберігання паролів»	26.04 – 04.05	Виконано
9.	Оформлення розділу «Практичне дослідження ефективності алгоритмів хешування»	05.05 – 14.05	Виконано
10.	Виконання завдання до підрозділу «Безпека життєдіяльності, основи охорони праці»	15.05 – 21.05	Виконано
11.	Оформлення кваліфікаційної роботи	23.05 – 08.06	Виконано
12.	Нормоконтроль	10.06 – 12.06	Виконано
13.	Перевірка на плагіат	12.06 – 14.06	Виконано
14.	Попередній захист кваліфікаційної роботи	19.06.2024	Виконано
15.	Захист кваліфікаційної роботи	26.06.2024	

Студент ____________ Шарик О.В.
(підпис) (прізвище та ініціали)

Керівник роботи ____________ Загородна Н. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дослідження сучасних алгоритмів хешування паролів користувачів // Кваліфікаційна робота ОР «Бакалавр» // Шарик Олександр Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБ-41 // Тернопіль, 2024 // с. – 56, рис. – 12, табл. – 5, лістинги – 7, бібліогр. – 15, додатки – 1.

Ключові слова: Криптографія, Хеш-Функція, Bcrypt, Scrypt, Argon2, Pbkdf2, Вразливості, Хешування, Автентифікація, Цілісність інформації, Захист Інформації, Безпека Паролів.

Кваліфікаційна робота присвячена дослідженню сучасних алгоритмів хешування паролів користувачів та оцінки їх ефективності. З метою забезпечення безпеки інформаційних систем, вибір надійного алгоритму хешування є критично важливим тому, що від цього залежить захист конфіденційних даних. У роботі розглянуто чотири популярні алгоритми хешування: bcrypt, scrypt, Argon2 та PBKDF2.

У процесі дослідження було проведено теоретичний аналіз алгоритмів, де розглянуто їх архітектуру, механізми захисту від атак та основні параметри конфігурації. Також було проведено експериментальні дослідження для оцінки продуктивності алгоритмів з точки зору часу виконання, використання пам'яті та завантаження процесора.

Розроблений підхід та отримані результати можуть бути використані адміністраторами систем та розробниками для покращення захисту паролів у різних інформаційних системах. Кваліфікаційна робота також може бути корисною для студентів та фахівців у сфері кібербезпеки, які бажають поглибити свої знання про сучасні криптографічні алгоритми та їх застосування для захисту паролів.

ANNOTATION

Research on Modern User Password Hashing Algorithms // Qualification Work for the Bachelor's Degree // Sharyk Oleksandr Volodymyrovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, Group SB-41 // Ternopil, 2024 // p. – 56, fig. – 12, tab. – 5, listings. – 7, bibliography. – 15, added. – 1.

Keywords: Cryptography, Hash Function, Bcrypt, Scrypt, Argon2, Pbkdf2, Vulnerabilities, Hashing, Authentication, Information Integrity, Information Protection, Password Security.

The qualification work is dedicated to the study of modern user password hashing algorithms and the evaluation of their effectiveness. To ensure the security of information systems, selecting a reliable hashing algorithm is critically important because it determines the protection of confidential data. The work examines four popular hashing algorithms: bcrypt, scrypt, Argon2, and PBKDF2.

During the research, a theoretical analysis of the algorithms was conducted, considering their architecture, protection mechanisms against attacks, and key configuration parameters. Experimental studies were also performed to evaluate the performance of the algorithms in terms of execution time, memory usage, and CPU load.

The developed approach and obtained results can be used by system administrators and developers to improve password protection in various information systems. The qualification work can also be useful for students and professionals in the field of cybersecurity who wish to deepen their knowledge of modern cryptographic algorithms and their application for password protection.

ЗМІСТ

ВСТУП.. 8
1 АНАЛІЗ ПРОБЛЕМИ ЗБЕРЕЖЕННЯ ПАРОЛІВ КОРИСТУВАЧІВ............... 10
1.1 Особливості автентифікації на основі паролів 10
1.2 Огляд атак на паролі користувачів.. 12
2 АНАЛІЗ СУЧАСНИХ АЛГОРИТМІВ ХЕШУВАННЯ ДЛЯ ЗБЕРІГАННЯ ПАРОЛІВ 18
2.1 Аналіз характеристик алгоритму хешування bcrypt.................................... 18
2.2 Аналітичний аналіз властивостей алгоритму хешування PBKDF2............ 20
2.3 Аналіз характеристик алгоритму хешування для паролів користувачів scrypt 25
2.4 Аналіз властивостей алгоритму хешування паролів користувачів argon2 28
3 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ХЕШУВАННЯ... 33
3.1 Характеристика пристрою та середовища використаних для дослідження 33
3.2 Розробка програм для дослідження обраних алгоритмів хешування......... 35
3.3 Тестування та порівняння ефективності досліджуваних алгоритмів хешування 40
4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ 46
4.1 Працездатність людини-оператора 46
4.2 Вимоги до режимів праці і відпочинку при роботі з ВДТ........................... 48
4.3 Системи протипожежної безпеки в ЦОД 50
ВИСНОВКИ.. 52
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 53
ДОДАТОК.. 55
Додаток А Код програми, що реалізує заміри ефективності алгоритмів хешування 55

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HMAC – hash-based message authentication code

NIST – National Institute of Standards and Technology

OWASP – Open Web Application Security Project

PBKDF2 – Password-Based Key Derivation Function 2

SHA – Secure Hash Algorithm

CPU – central processing unit (центральний процесор)

ЕОМ – електронно-обчислювальна машина

ВДТ – візуальний дисплей термінал

ЦОД – центр опрацювання даних

ВСТУП

У сучасному цифровому світі, де особиста інформація стає все більш важливою та чутливою, безпека збереження паролів користувачів є надзвичайно актуальною проблемою. Зловмисники постійно шукають способи доступу до цих даних, щоб використовувати їх у шкідливих цілях, таких як крадіжка особистої інформації, фінансові маніпуляції або порушення конфіденційності.

Алгоритми хешування паролів є важливою ланкою в системах безпеки, що забезпечують захист цих даних. Вибір відповідного алгоритму хешування має велике значення для забезпечення стійкості паролів до різних видів атак, а також забезпечення ефективного збереження та обробки паролів у системах.

Однак, з появою нових технологій та методів атак, необхідно постійно оновлювати та адаптувати алгоритми хешування паролів. Саме тому дослідження та аналіз сучасних алгоритмів хешування стає крайньо важливою задачею, яка спрямована на забезпечення найвищого рівня безпеки та захисту персональної інформації користувачів.

В рамках даної роботи ми прагнемо дослідити сучасні алгоритми хешування паролів, оцінити їхню ефективність, безпеку та придатність для практичного застосування. Це дослідження має на меті виявлення найкращих практик та рекомендацій для вибору алгоритмів хешування паролів у різних сценаріях застосування.

Таким чином, дана робота має велике значення для розуміння та підвищення рівня безпеки у сфері збереження паролів користувачів, що в свою чергу сприятиме забезпеченню конфіденційності в онлайн-середовищі.

Метою даної роботи є дослідження сучасних алгоритмів хешування паролів користувачів задля визначення їхньої ефективності, безпеки та придатності для практичного застосування. Конкретні цілі дослідження включають:

– опис проблеми збереження паролів;
– огляд сучасних алгоритмів хешування паролів, таких як scrypt, bcrypt, Argon2, PBKDF2;

– аналіз їх криптографічних властивостей;
– вимірювання ефективності, а саме визначення швидкодії та витрат ресурсів кожного алгоритму;
– здійснення порівняльного аналізу за характеристиками, такими як безпека, продуктивність та придатність для практичного застосування.

Предметом дослідження є сучасні алгоритми хешування паролів, які використовуються для збереження та обробки парольних даних в інформаційних системах. Ці алгоритми є ключовою складовою в системах безпеки, оскільки вони забезпечують захист паролів користувачів від несанкціонованого доступу та зловмисницьких атак.

Об'єктом дослідження є самі алгоритми хешування паролів, а також їхні характеристики, властивості та ефективність у різних умовах застосування.

1 АНАЛІЗ ПРОБЛЕМИ ЗБЕРЕЖЕННЯ ПАРОЛІВ КОРИСТУВАЧІВ

1.1 Особливості автентифікації на основі паролів

Завдання захисту від несанкціонованого доступу та захист персональних даних вирішує автентифікація. Автентифікація – це процес перевірки достовірності особи або системи, що намагається отримати доступ до певних ресурсів або інформації на основі ідентифікатора. Існують різні типи методів автентифікації, які можуть бути використані для веб-додатків, щоб забезпечити безпеку як самого сайту, так і його користувачів. Варто зазначити, що автентифікація використовується повсюди, не тільки у веб-додатках. Існує декілька видів автентифікації, а саме: автентифікація на основі пароля, багатофакторна автентифікація, автентифікація на основі сертифікату, біометрична автентифікація та автентифікація на основі токенів. Розглянемо детальніше саме автентифікацію на основі пароля.

Автентифікація на основі паролів (або ж однофакторна автентифікація) – це метод автентифікації, який вимагає від користувача створення пароля для свого облікового запису, після чого пароль хешується за допомогою алгоритмів хешування. Після чого хешований пароль зберігається в базі даних, щоб у випадку її компрометації хакер не міг визначити правильний пароль. Коли користувач хоче увійти, він вводить свої облікові дані, і якщо вони правильні, користувач автентифікується [1].

Отож, зберігання паролів користувачів є критично важливим компонентом будь-якої програми чи веб-додатку. Під час зберігання паролю користувача, важливо переконатися, що він захищений таким чином, щоб у разі зламу зловмисники не могли розкрити пароль користувача.

Багато випадків витоку даних користувачів, які викликали гучні реакції громадськості, підкреслюють критичну необхідність захисту паролів. У більшості випадків, коли веб-сайти або веб-додатки стають жертвами кібератак, одним із головних об'єктів атак є саме база даних з паролями користувачів.

Ситуація стає ще більш небезпечною, коли розробники веб-ресурсу не застосовують належні заходи безпеки для збереження паролів користувачів.

Під час розробки веб-сайту або веб-програми, яка потребує збереження даних користувачів, критично важливо приділити належну увагу заходам безпеки. Використання безпечних методів зберігання паролів, таких як сучасні алгоритми хешування та зберігання з унікальними "солями" для кожного паролю, є невід'ємною частиною цього процесу. Крім того, необхідно вжити заходів для захисту бази даних, щоб запобігти несанкціонованому доступу до паролів користувачів.

Враховуючи постійну еволюцію методів кібератак та підвищення їхньої складності, збереження паролів користувачів у безпечному форматі залишається завданням першочергового значення для розробників веб-додатків. Захищені паролі не лише забезпечують безпеку індивідуальних облікових записів, але й сприяють підвищенню довіри та репутації веб-ресурсів і додатків серед користувачів.

Хеш-функція – це функція, яка призначена для перетворення довільного повідомлення або набору даних, які зазвичай представлені у бітовому вигляді, в одну фіксовану бітову комбінацію певної довжини. Ця вихідна комбінація часто називається тегом, згорткою, дайджестом або відбитком.

Хешування – це процес використання хеш-функції для перетворення вхідних даних будь-якого розміру в вихідну послідовність фіксованої довжини. Процес хешування можна відобразити схемою, як на рисунку 1.1.

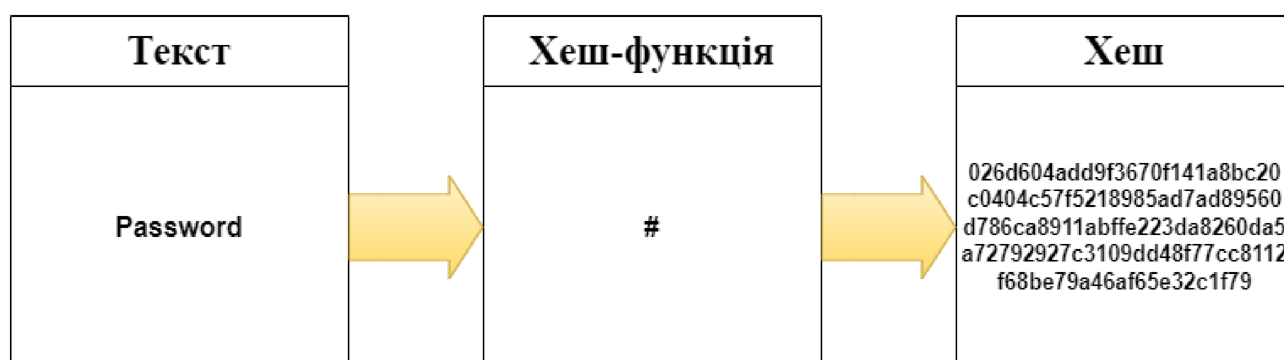

Рисунок 1.1 – Схема хешування

Хешування та шифрування можуть забезпечити захист конфіденційних даних, проте в більшості випадків паролі слід хешувати, а не шифрувати. Оскільки хешування є односторонньою функцією, а це означає, що неможливо зворотно перетворити хеш у вихідне значення відкритого тексту, це є найбільш прийнятним методом для зберігання та перевірки пароля. Навіть у випадку отримання зловмисником хешованого пароля, він не зможе використати його для входу як жертва.

Натомість шифрування, будучи двосторонньою функцією, може дати можливість зловмисникам отримати вихідний відкритий текст з зашифрованих даних.

Отож, одна з переваг хешування є одностороннім процесом. Крім того, для будь-яких вхідних даних генерується унікальне хеш-значення. Це дозволяє впевнитися в тому, що різні вхідні дані в результаті будуть мати різні хеші. Ще одною особливістю хешів є фіксована довжина. Хеш-функції генерують вихідні значення фіксованої довжини, незалежно від розміру вхідних даних. Це дозволяє зберігати хеші в структурах даних фіксованого розміру.

1.2 Огляд атак на паролі користувачів

Надійні паролі, які зберігаються за допомогою сучасних алгоритмів хешування та найкращих практик хешування, повинні бути фактично недосяжними для зловмисника. Однак існують ситуації, коли зловмисник може "зламати" хеші за певних обставин. Існує декілька поширених видів атак спрямованих на хеші, такі як: атака "грубої сили” (методом перебору), атака за допомогою словника, атака з використанням колізій та атака веселкової таблиці (Rainbow Table Attack).

Атака за допомогою словника є досить примітивною та затратною, крім того вона працює лише тоді, коли користувач використав слабкий пароль такий, як "password". Користувачі схильні вибирати легкі для запам'ятовування та передбачувані паролі. Витоки паролів показують, що вони використовують поширені імена та прізвища, імена домашніх тварин, назви груп, назви

спортивних команд тощо. Ці фрази комбінуються разом з паролями, отриманими з останніх витоків, щоб створити список паролів. Тоді зловмисники замість того, щоб пробувати всі можливі комбінації символів, використовують слова з цих словників для злому паролів. У 2023 році компанією NordPass, яка надає рішення по створенню паролів і безпечного керування ними, було проведено дослідження про найбільш поширені паролі. В результаті компанія опублікувала 200 паролів, які набули широкого використання, десять найпоширеніших наведено в таблиці 1.1.

Таблиця 1.1 – Найбільш поширені паролі

Номер в рейтингу	Пароль	Кількість використань
1	123456	4,524,867
2	admin	4,008,850
3	12345678	1,371,152
4	123456789	1,213,047
5	1234	969,811
6	12345	728,414
7	password	710,321
8	123	528,086
9	Aa123456	319,725
10	1234567890	302,709

Це дослідження вкотре підтверджує, що важливим аспектом захисту персональних даних є не тільки процес зберігання паролів за допомогою сучасних хеш-функцій, а й складність самих паролів.

Атаки методом перебору зазвичай застосовуються до хешованих значень паролів для отримання оригінального відкритого тексту. Ці атаки є інтенсивними для центрального процесора і тому займають багато часу. У атаках методом перебору зловмисник пробує кожну можливу комбінацію доступних символів. Наприклад, якщо атакувати пароль із восьми символів, атака почнеться з "aaaaaaaa" і поступово збільшуватиметься до "zzzzzzzz". Також існують різні

конфігурації для налаштування набору символів на основі даних. Зловмисник може вибрати набір символів, наприклад, лише числові, алфавітно-цифрові тощо [2].

Через властивість хешування зловмисники не можуть зворотно обчислити хеш, щоб отримати пароль. Однак, зловмисники можуть використовувати попередньо обчислені таблиці для пошуку хешів. Ці таблиці називаються веселковим таблицями. Коли зловмисник має доступ до бази даних, де паролі зберігаються в хешованому вигляді замість відкритого тексту, він може використовувати веселкові таблиці для пошуку хеш-значень і знаходження оригінальних паролів. Проте, вони ефективні лише проти хешів без солі, оскільки додавання випадкової солі до кожного пароля робить кожен хеш унікальним. Це значно збільшує кількість необхідних обчислень і робить використання веселкових таблиць непрактичним.

Ще одним видом атак, є атаки на колізії першого та другого роду. Колізії першого роду – це ситуації, коли два різні вхідні повідомлення або дані мають однаковий хеш-значення після застосування хеш-функції тобто, коли два різних вхідних повідомлення породжують той самий хеш. Це відбувається через обмежену довжину хеш-значень та нескінченну кількість можливих вхідних даних.

Колізії другого роду, також відомі як другорядні колізії, є поняттям з області криптографії та безпеки даних, яке виникає в контексті атак на хеш-функції. Колізії другого роду стосуються ситуацій, коли атакуючий може спеціально створити два різних вхідних повідомлення, які генерують однаковий хеш.

У відміну від колізій першого роду, які можуть бути отримані шляхом випадкового вибору даних або за допомогою спеціальних атак, колізії другого роду потребують більш складного підходу та обчислювальних зусиль. А також вони є показником вразливості хеш-функцій.

Саме завдяки високошвидкісному обладнанню, наприклад графічним процесорам і хмарним сервісам з великою кількістю серверів для оренди,

вартість успішного злому пароля для зловмисника відносно невелика, особливо якщо не дотримуватися найкращих практик хешування та створення паролів.

1.3 Методи пом’якшення та усунення впливу від атак на паролі користувачів

Існують методи покращення зберігання паролі, а саме засолювання, поперчення та використання робочих факторів, всі ці методи зменшують ймовірність того, що зловмисник отримає паролі у вигляді тексту.

Сіль – це унікальний, випадково згенерований рядок, який додається до кожного пароля під час процесу хешування [3]. Оскільки сіль унікальна для кожного користувача, зловмисник повинен розшифрувати кожен хеш один за одним за допомогою відповідної солі, а не обчислювати хеш один раз і порівнювати його з усіма збереженими хешами. Це значно ускладнює злам великої кількості хешів, оскільки необхідний час збільшується прямо пропорційно кількості хешів. Соління також захищає від попереднього обчислення хешу або пошуку на основі бази даних. Крім того, якщо пароль однаковий, різні солі приводять до різних хешів, тому неможливо визначити, чи є у двох користувачів однаковий пароль, схему роботи соління паролів зображено на рисунку 1.2.

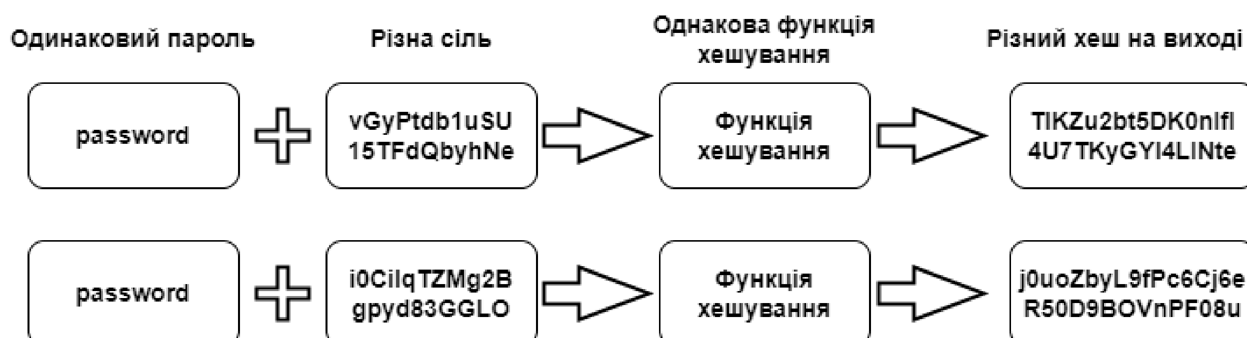

Рисунок 1.2 – Схема роботи соління паролів

Сучасні алгоритми хешування, такі як Argon2id, bcrypt і PBKDF2, автоматично підсолюють паролі, тому під час їх використання не потрібні додаткові дії.

На додаток до соління можна використовувати перець, щоб забезпечити додатковий рівень захисту. Він не дає зловмиснику зламати будь-який з хешів, якщо у нього є доступ тільки до бази даних, наприклад, якщо він скористався уразливістю SQL-ін'єкції або отримав резервну копію бази даних. Стратегії перебору жодним чином не впливають на функцію хешування паролів.

Наприклад, одна зі стратегій використання перцю полягає в хешуванні паролів як зазвичай (за допомогою алгоритму хешування паролів), а потім у використанні HMAC (hash-based message authentication code). HMAC – це криптографічний метод, який використовує хеш-функцію разом із секретним ключем для забезпечення автентичності та цілісності повідомлення. HMAC використовує стандартні хеш-функції, такі як SHA-256, SHA-1 або інші [3]. Хеш-функція застосовується до паролю разом із секретним ключем. Процес генерації HMAC складається з кількох кроків. Спершу, якщо довжина секретного ключа менша за блоковий розмір хеш-функції, ключ доповнюється нулями до потрібної довжини. Якщо він довший, то хешується і результат використовується як ключ. Далі генеруються два різні ключі шляхом XOR-операції секретного ключа з двома різними константами (опорними блоками), внутрішнім опорним блоком і зовнішнім опорним блоком. Після чого внутрішній ключ об'єднується з повідомленням, і результат хешується, а зовнішній ключ об'єднується з результатом внутрішнього хешування, і цей результат знову хешується. Останній хеш є HMAC для паролю. Перець є спільним для всіх збережених паролів, а не унікальним, як у випадку з сіллю. На відміну від солі пароля, перець не повинен зберігатися в базі даних.

Ще одним методом є використанням робочих елементів. Коефіцієнт роботи – це кількість ітерацій алгоритму хешування, які виконуються для кожного пароля. Зазвичай коефіцієнт роботи зберігається в хеш-виводі. Це збільшує витрати на обчислення хешу, що, у свою чергу, зменшує швидкість

обчислень, однак збільшує ресурси, які зловмисник має потратити, щоб спробувати зламати хеш пароля.

Важливо знайти баланс між безпекою та продуктивністю під час вибору робочого фактора. Незважаючи на те, що додаткові робочі фактори роблять хеші складнішими для зловмисника, вони також сповільнюють процес перевірки спроби входу в систему. Якщо робочий фактор занадто високий, продуктивність програми може погіршитися. Зловмисники можуть використати це для проведення атаки на відмову в обслуговуванні, виснажуючи центральний процесор сервера великою кількістю спроб входу. Наявність робочого фактору має велику перевагу, яка полягає в тому, що він може бути збільшеним з часом, оскільки апаратне забезпечення стає дешевшим і потужнішим. Найпростіший спосіб оновити робочий фактор – дочекатися наступної автентифікації користувача, а потім перехешувати пароль.

Отже, при дотримуванні вище наведених практик можна забезпечити найвищий рівень безпеки паролів користувачів, ускладнюючи їхній злам та збільшуючи витрати обчислювальних ресурсів для зловмисників.

2 АНАЛІЗ СУЧАСНИХ АЛГОРИТМІВ ХЕШУВАННЯ ДЛЯ ЗБЕРІГАННЯ ПАРОЛІВ

Деякі сучасні алгоритми хешування були спеціально розроблені для безпечного зберігання паролів. В роботі проводиться глибокий аналіз та практичне дослідження ефективності чотирьох найбільш використовуваних алгоритмів хешування, а саме: bcrypt, scrypt, Argon2 та PBKDF2. Всі чотири алгоритми рекомендовані організацією OWASP (Open Web Application Security Project). Алгоритмом хешування PBKDF2, рекомендований NIST і має реалізацію, затверджену стандартом FIPS-140 (урядовий стандарт комп'ютерної безпеки США, який використовується для схвалення криптографічних модулів). Argon2 став переможцем конкурсу з хешування паролів 2015 року. Саме тому в межах дослідження були обрані ці чотири алгоритми.

2.1 Аналіз характеристик алгоритму хешування bcrypt

Bcrypt – це хеш-функція, розроблена Нільсом Провосом і Девідом Мазіерсом для хешування паролів на основі шифрування Blowfish. Її було розроблено спеціально, щоб підвищити надійність зберігання паролів у системі автентифікації Unix. Також bcrypt може забезпечувати надійний захист паролів від атак перебором, завдяки чому він широко використовується у різних додатках та системах для зберігання паролів. Алгоритм хешування bcrypt адаптивний до обчислювальних потужностей апаратного забезпечення. Одним із параметрів, який Bcrypt отримує, є кількість ітерацій. Можна налаштувати кількість ітерацій до певного рівня, що сповільнює процес злому з точки зору безпеки, але є достатньо швидким, щоб перевірити заданий пароль. Bcrypt використовує переваги дорогого налаштування ключів в Eksblowfish. Eksblowfish – це дороговартісна варіація блочного шифру Blowfish з розкладом ключів Blowfish. Eksblowfish приймає три параметри: ключ, сіль і вартість. Цей алгоритм є адаптивним до обчислювальних потужностей завдяки параметру

вартості. Розклад ключів стає дорожчим, коли значення зростає. Користувач вибирає пароль, який є головним параметром.

Використовуючи 128-бітову сіль, Bcrypt може шифрувати 192-бітові хеші. Значення хешу складаються з 192 біт (основа-64, закодована як 31 символ), а значень солі складаються з 128 біт (основа-64, закодована як 22 символи). На вході користувач повинен ввести пароль довжиною не більше 72 байт, інакше пароль буде урізано. Eksblowfish повертає набір підключів та S-блоки. Нижче наведено етапи роботи алгоритму Eksblowfish:

- Спочатку відбувається копіювання цифр числа π спочатку у підключі, а потім копіювання їх у S-блоки.
- Розширення ключа шляхом модифікації P-масиву та S-блоків на основі значення 128-бітної солі та ключа змінної довжини. Над усіма підключами в P-масиві виконується операція XOR з ключем шифрування. А над усіма i-тими 32 бітами ключами виконується XOR з i-ми P-масиву.
- Шифрування ключа за допомогою алгоритму шифрування Blowfish протягом 2^{cost} разів [4].

Після отримання набору підключів та S-блоків від Eksblowfish, bcrypt шифрує пароль повторно 64 рази в режимі, який називається Electronic Codebook. Режим Electronic Codebook є одним із режимів роботи блочного шифру. Результат шифрування потім об'єднується з параметрами cost та salt для надання інформації для подальшого процесу перевірки. Bcrypt має наступну схему: $2b$[ітерація]$[сіль][значення хешу]. Для прикладу візьмемо значення паролю (password), тоді на виході ми отримаємо значення хешу: $2b$12$V/24rFb/sArw2kyGrtQ.9.08J2YBoX3bv7Ndz0k1ArNC4VUNcpHrK, розкладаючи його за схемою, можна виявити, що параметр складності рівний 12, що вказує на 2^{12} раундів розкладання ключа. Сіль в даному випадку буде рівна значенню: NUtreoOeM4so4iKU1oPJZO, а сам хеш паролю рівний значенню: gpAw/RqvBsOIAYsTPtpUi8REa8lIxcq. Схему захисту паролів за допомогою bcrypt наведено на рисунку 2.1.

Рисунок 2.1 – Схема хешування паролів за допомогою bcrypt

Bcrypt було реалізовано на багатьох мовах програмування мовами, такими як C, C++, C#, Go, Java, Javascript, Perl, PHP, Python, Ruby та інші мови. Спочатку він використовувався для системи автентифікації в OpenBSD, але тепер він набув широкого використання для безпечного зберігання паролів у багатьох веб-додатках. Завдяки своїй адаптивності, використанню солі та високій складності обробки, bcrypt забезпечує ефективний захист від атак типу грубої сили та атак із використанням обчислювальних ресурсів.

2.2 Аналітичний аналіз властивостей алгоритму хешування PBKDF2

Алгоритм PBKDF2 (Password-Based Key Derivation Function 2) є стандартом для генерації криптографічно стійких ключів на основі паролів [5]. Основна мета PBKDF2 – забезпечити надійний метод для отримання ключів з паролів, які зазвичай не мають високої ентропії (не дуже випадкові).

Алгоритм хешування PBKDF2 має декілька основних особливостей:

- Розтягування ключа (Key Stretching): PBKDF2 використовує багаторазове хешування пароля для підвищення обчислювальної складності підбору пароля шляхом атак типу грубої сили. Це робиться за допомогою численних ітерацій хешування, що значно збільшує час, необхідний для кожної спроби підбору пароля.
- Використання солі (Salt):Для кожного пароля генерується унікальна сіль (salt), яка додається до пароля перед хешуванням. Це запобігає створенню веселкових таблиць (precomputed hash tables) для швидкого пошуку паролів.
- Налаштовуваність: Кількість ітерацій, довжина солі та хеш-функція можуть бути налаштовані залежно від вимог до безпеки та продуктивності. Зазвичай рекомендується використовувати значну кількість ітерацій для підвищення стійкості до атак.

Алгоритм PBKDF2 вимагає, щоб був вибраний внутрішній алгоритм хешування, такий як HMAC або ряд інших алгоритмів хешування. Наприклад HMAC-SHA-256 широко підтримується і рекомендований NIST. Робочий фактор для PBKDF2 реалізується через кількість ітерацій, яка повинна бути встановлена по-різному в залежності від використовуваного алгоритму внутрішнього хешування:

- PBKDF2-HMAC-SHA1: 1 300 000 ітерацій.
- PBKDF2-HMAC-SHA256: 600 000 ітерацій.
- PBKDF2-HMAC-SHA512: 210 000 ітерацій [3].

Хорошою практикою при реалізації PBKDF2 є попереднє хешування. Якщо PBKDF2 використовується з HMAC і пароль довший за розмір блоку хеш-функції (64 байти для SHA-256), пароль автоматично попередньо хешується.

Принцип роботи алгоритму PBKDF2 полягає в наступному, функціонує шляхом повторного застосування криптографічної хеш-функції до комбінації пароля та солі. Детальний процес роботи PBKDF2 можна розбити на чотири умовних кроки: ініціалізація, обчислення блоків, комбінація блоків та формування результату.

Ініціалізація є початковим етапом, на якому задаються основні параметри такі, як: пароль, сіль, кількість ітерацій (від 210 000 до 1 300 000) та довжина кінцевого ключа.

Другим етапом є обчислення блоків. Ключ генерується блоками. Кожен блок обчислюється шляхом багаторазового застосування хеш-функції до комбінації пароля, солі та номера блоку. Наприклад, для обчислення першого блоку H1 використовується наступна формула (2.1):

$$H1 = F(\text{Пароль}, \text{Сіль}, \text{Ітерації}, 1) \tag{2.1}$$

Де значення F(Пароль, Сіль, Ітерації, 1) обчислюється за формулою (2.2), та формулами (2.3) і (2.4):

$$U_1 = PRF(\text{Пароль}, \text{Сіль} \parallel 1) \tag{2.2}$$

$$U_2 = PRF(\text{Пароль}, U_1) \tag{2.3}$$

$$U_i = PRF(\text{Пароль}, U_{i-1}) \tag{2.4}$$

В наведених формулах PRF (Pseudorandom Function) є криптографічною хеш-функцією, зазвичай HMAC (Hash-based Message Authentication Code).

Після проведених ітерацій результат обчислюється за формулою (2.5):

$$H1 = U_1 \oplus U_2 \oplus U_3 \oplus \ldots \oplus U_i \tag{2.5}$$

Аналогічно обчислюються інші блоки H2, H3, ..., Hn до отримання потрібної довжини ключа.

Наступний етапом є комбінація блоків. Всі блоки комбінуються разом для створення кінцевого ключа заданої довжини.

І в фінальному етапі формується результат. Отриманий ключ є результатом роботи PBKDF2, який може бути використаний для різних криптографічних цілей, таких як шифрування, автентифікація або зберігання паролів.

Візьмемо приклад і розглянемо роботи PBKDF2 з конкретними параметрами:

– Пароль: "password"
– Сіль: "salt"
– Кількість ітерацій: 1000
– Довжина ключа: 256 біт (32 байти)
– Хеш-функція: HMAC-SHA-256

Пароль, сіль і кількість ітерацій задаються. Хеш-функція HMAC-SHA-256 обирається для генерації псевдовипадкових даних. Обчислення першого блоку можна описати формулою (2.6):

$$U_1 = HMAC - SHA - 256("password", "salt" \parallel 1) \tag{2.6}$$

Повторимо ітерації, як в формулі (2.7) та (2.8).

$$U_2 = HMAC - SHA - 256("password", "salt", U_1) \tag{2.7}$$

$$U_{1000} = HMAC - SHA - 256("password", "salt", U_{999}) \tag{2.8}$$

Після завершення 1000 ітерацій результат обчислюється за формулою (2.9):

$$H1 = U_1 \oplus U_2 \oplus U_3 \oplus \ldots \oplus U_{1000} \tag{2.9}$$

Аналогічним чином обчислюються блоки H2, H3, ..., Hn до отримання потрібної довжини ключа. Після чого всі обчислені блоки комбінуються для отримання кінцевого ключа заданої довжини. На виході буде отримане хеш-значення, як на рисунку 2.3.

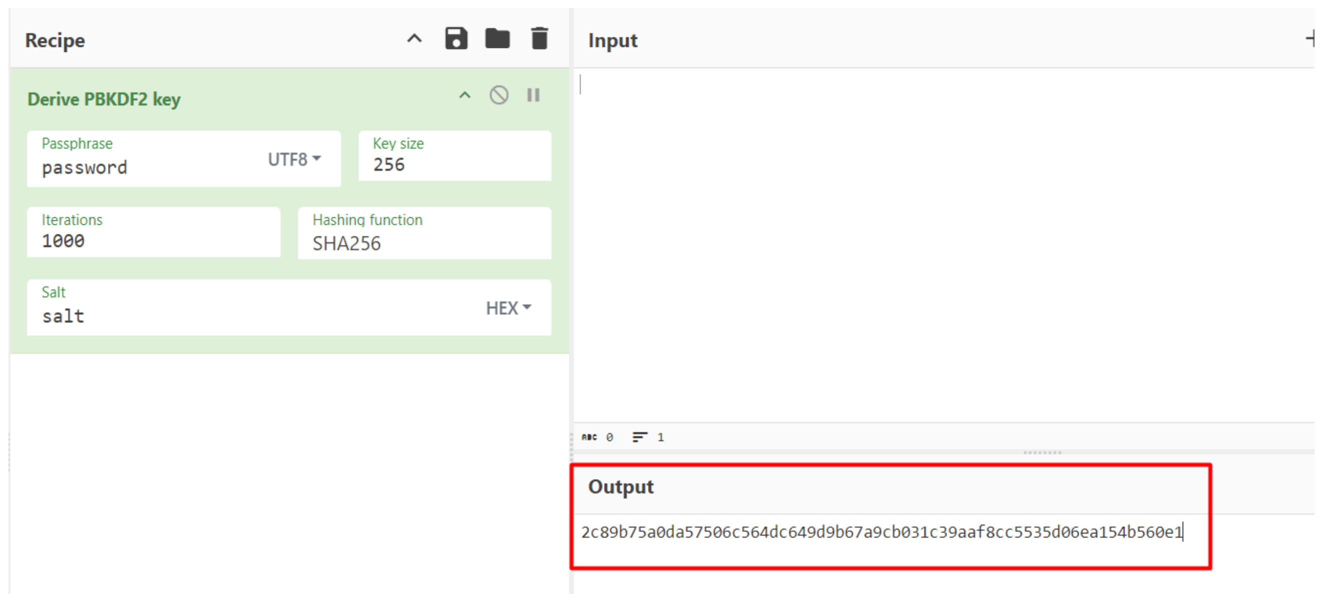

Рисунок 2.2 – Хеш-значення отримане алгоритмом PBKDF2

Існує декілька переваг використання алгоритму хешування. Нижче наведені декілька з них:

– Завдяки використанню багаторазового хешування та солі, PBKDF2 значно ускладнює проведення атак типу грубої сили.
– Для однакових вхідних даних та параметрів PBKDF2 завжди генерує один і той самий вихідний ключ.
– Кількість ітерацій може бути налаштована для досягнення бажаного рівня безпеки, зважаючи на доступні обчислювальні ресурси.
– Сумісність: PBKDF2 є стандартним алгоритмом, що широко використовується в різних криптографічних протоколах та системах.

Найбільше алгоритм PBKDF2 він використовується для безпечного зберігання хешів паролів. Навіть якщо база даних з хешами буде скомпрометована, зловмисникам буде дуже складно відновити оригінальні паролі завдяки використанню солі та багаторазового хешування. Крім цього, PBKDF2 застосовується для генерації криптографічно стійких ключів з паролів або інших секретних фраз, що використовуються в системах шифрування та автентифікації. Також алгоритм використовується в різних безпечних протоколах та системах, таких як WPA2 (для захисту бездротових мереж), PGP (для шифрування електронної пошти) та інших.

У підсумку, PBKDF2 є надійним і добре дослідженим алгоритмом, що забезпечує високий рівень безпеки при хешуванні паролів та генерації ключів. Використання солі та багаторазове хешування робить цей алгоритм стійким до атак типу грубої сили та веселкових таблиць. PBKDF2 активно використовується в багатьох сучасних системах для захисту паролів та генерації криптографічних ключів. Завдяки його надійності та гнучкості, він залишається одним з найпопулярніших методів захисту паролів у сучасній криптографії.

2.3 Аналіз характеристик алгоритму хешування для паролів користувачів scrypt

Алгоритм хешування scrypt був розроблений Коліном Персівалем у 2009 році для використання в проекті Tarsnap (сервіс захищеного зберігання резервних копій). Основна мета scrypt полягає у створенні алгоритму хешування, який був би стійкішим до атак за допомогою апаратних засобів.

Існує декілька особливостей даного алгоритму хешування, що стосуються безпеки та стійкості. З основного це використання солі, через що він є стійким до різних видів атак. Іншою особливістю є висока вимога до пам'яті. Завдяки використанню великого обсягу пам'яті під час обробки блочного масиву, scrypt значно ускладнює паралельне виконання та апаратне прискорення. А також є можливість налаштування параметрів, що дозволяє балансувати між продуктивністю та безпекою, забезпечуючи оптимальний рівень захисту залежно від конкретних вимог і ресурсів.

Алгоритм хешування scrypt набув доволі широкого застосування. Найбільше він використовується в захисті паролів, а саме для безпечного зберігання хешів паролів у різних системах і додатках, забезпечуючи високий рівень захисту від атак типу грубої сили. Його також використовують для генерації криптографічно стійких ключів на основі паролів або інших секретних фраз. А також scrypt застосовується у деяких криптовалютах (наприклад, Litecoin), як основний алгоритм майнінгу, забезпечуючи стійкість до

спеціалізованого апаратного забезпечення та забезпечуючи децентралізованість майнінгу [6].

Алгоритм scrypt має три різні параметри, які можна налаштувати: мінімальний параметр витрат процесора/пам'яті (N), розмір блоку (r) і ступінь паралелізму (p) [3]. OWASP рекомендує використовувати одне з наступних налаштувань наведене в таблиці 2.1:

Таблиця 2.1 – Рекомендовані налаштування

N (MiB)	r	p
128	8 (1024 байти)	1
64	8 (1024 байти)	2
32	8 (1024 байти)	3
16	8 (1024 байти)	5
8	8 (1024 байти)	10

Ці налаштування конфігурації забезпечують однаковий рівень захисту. Єдиною відмінністю є компроміс між використанням процесора та оперативної пам'яті.

Принцип роботи алгоритму хешування scrypt складається з декількох основних етапів: ініціалізація параметрів, генерація псевдовипадкових даних, заповнення блочного масиву, змішування блочного масиву та фінальне хешування. Ініціалізація параметрів відбувається наступним чином:

- Пароль (P): вхідний пароль, який потрібно захистити.
- Сіль (S): випадковий вектор, який додається до пароля для забезпечення унікальності хешів навіть для однакових паролів.
- Параметри конфігурації: параметр N, що визначає обсяг пам'яті (число блоків пам'яті), параметр r, що визначає використання центрального процесора (CPUs) та параметр p, що визначає ступінь паралелізму.
- Довжина ключа (dkLen): бажана довжина вихідного ключа.

Наступним етапом є генерація псевдовипадкових даних: За допомогою функції PBKDF2-HMAC-SHA256 для створення псевдовипадкових даних на

основі пароля та солі. Цей етап забезпечує первинне хешування та підготовку даних для подальшої обробки.

Після цього відбувається заповнення блочного масиву (Block Array Filling). Створюється великий блочний масив розміром N*128*r байт. Масив заповнюється псевдовипадковими даними, генерованими на попередньому етапі. Під час цього процесу використовуються криптографічні примітиви для генерації та заповнення блочних даних.

Наступний етап це змішування та обробка блочного масиву (Block Array Mixing). Виконується багаторазова обробка блочного масиву, що включає читання, запис та хешування даних. Цей етап використовує велику кількість пам'яті та обчислювальних ресурсів, що робить процес складним для апаратного прискорення. Алгоритм обробки гарантує, що кожен блок пам'яті впливає на результат, що ускладнює паралельне виконання та оптимізацію на спеціалізованому апаратному забезпеченні.

Останнім етапом є фінальне хешування (Final Hashing). Після завершення обробки блочного масиву отримані дані знову хешуються за допомогою PBKDF2-HMAC-SHA256. Фінальне хешування забезпечує створення криптографічно стійкого ключа необхідної довжини (dkLen). В результаті якого ми отримуємо хеш-значення, як на рисунку 2.3.

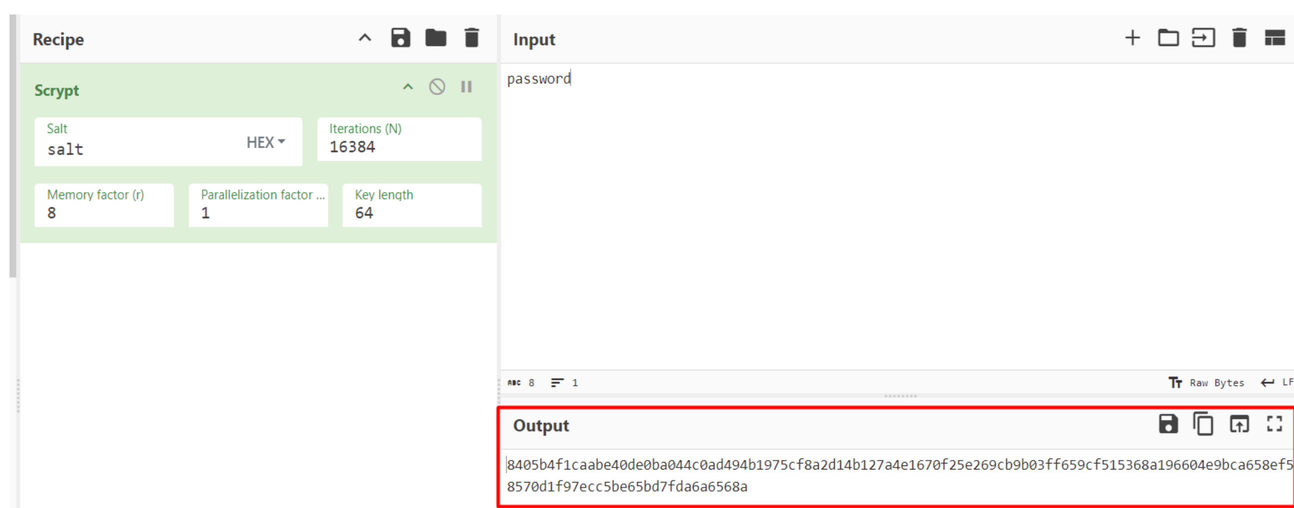

Рисунок 2.3 – Хеш-значення отримане алгоритмом scrypt

Отже, алгоритм scrypt є потужним інструментом для захисту паролів та генерації криптографічних ключів, що забезпечує високу стійкість до атак на основі обчислювальних ресурсів. Завдяки використанню великих обсягів пам'яті та налаштовуваних параметрів, scrypt є ефективним засобом для захисту даних у сучасних системах безпеки.

2.4 Аналіз властивостей алгоритму хешування паролів користувачів argon2

Argon2 – це сучасний алгоритм хешування паролів, розроблений спеціально для забезпечення високого рівня безпеки та стійкості до атак. Він був розроблений командою з Лювенського католицького університету під керівництвом Алекса Бірюкова, Деніеля Д'юка, Йоанни Крамарек та Дмитра Ховатова. У 2015 році Argon2 був оголошений переможцем конкурсу Password Hashing Competition (PHC) як найкращий алгоритм хешування паролів. Argon2 має кілька конфігурованих параметрів, які дозволяють налаштовувати використання процесора та пам'яті. Існує три варіанти Argon2: Argon2d, Argon2i та Argon2id, кожен з яких має свої особливості:

- Argon2d максимізує залежність від даних, щоб підвищити стійкість до паралельного апаратного забезпечення, такого як GPU. Однак цей варіант може бути вразливим до атак на основі побічних каналів.
- Argon2i, навпаки, розроблений для протидії атакам на основі побічних каналів, але може втратити частину стійкості до паралельного апаратного забезпечення.
- Argon2id поєднує обидва підходи: спочатку працює як Argon2i, а потім як Argon2d. Це гібрид, що об'єднує переваги обох варіантів і наразі є найбільш рекомендованим [7].

Однією з найважливіших особливостей Argon2 є його здатність оптимізувати одночасно використання процесора і пам'яті, що підвищує вартість для зловмисників, які намагаються здійснити атаки методом перебору для вгадування пароля. Крім того, Argon2 використовує унікальні солі, що запобігає колізіям хешів серед користувачів, які використовують однакові паролі Ця

характеристика є важливою для запобігання атакам за допомогою попередньо обчислених хешів, таких як веселкові таблиці.

Argon2 має три різні параметри, які можна налаштувати: базовий мінімальний розмір пам'яті (m), мінімальна кількість ітерацій (t) і ступінь паралелізму (p) [3]. OWASP (Open Web Application Security Project) рекомендує використовувати наступні параметри конфігурації, наведені в таблиці 2.1:

Таблиця 2.1 – Параметри конфігурації

M (MiB)	t	p
46	1	1
19	2	1
12	3	1
9	4	1
7	5	1

Внутрішня структура алгоритму хешування Argon2 складається з декількох етапів:

- Ініціалізація: алгоритм спочатку створює початковий блок, використовуючи вхідні значення від користувача, такі як пароль, сіль і, за бажанням, секретні дані. Цей початковий блок заповнює послідовність блоків, використовуючи пам'ять.
- Заповнення блоків: потім алгоритм заповнює ці блоки пам'яті до встановленого обсягу пам'яті. Обчислення кожного блоку залежить від попередніх блоків, що робить цей етап залежним від даних. Argon2d і Argon2i використовують різні підходи під час цього процесу заповнення блоків.
- Створення фінального блоку: після заповнення всіх блоків алгоритм вибирає один блок як фінальний.
- Генерація хеша: нарешті, алгоритм передає цей фінальний блок до хеш-функції для створення кінцевого хеша пароля.

Схему алгоритму хешування Argon2 зображено на рисунку 2.4.

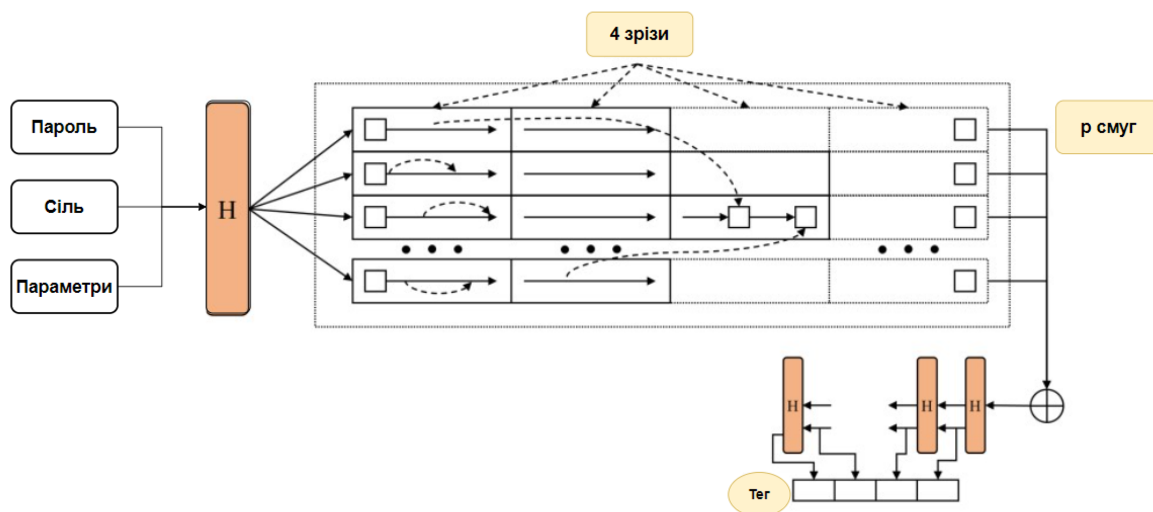

Рисунок 2.4 – Схема однопрохідного Argon2 з p смугами і 4 зрізами

Ці етапи працюють разом, щоб забезпечити безпеку та міцність процесу хешування паролів у Argon2. Використовуючи операції, що використовують багато пам’яті та залежність від даних, Argon2 має на меті зробити спроби злому паролів більш дорогими та тривалими.

Алгоритм Argon2 має велику залежність від даних, що ускладнює атаки методом перебору або атаки, засновані на GPU. Завдяки цьому Argon2 широко визнається як надійний алгоритм хешування паролів.

Вхідні параметри, які використовуються в алгоритмі Argon2, можна перерахувати наступним чином:

- Пароль: пароль користувача, що використовується для генерації зашифрованого хеш-значення.
- Сіль: випадково згенероване значення. Сіль додається до пароля кожного користувача для покращення процесу шифрування. Це гарантує, що користувачі з однаковими паролями не матимуть однакових хеш-значень.
- Витрата часу: параметр, що регулює кількість часу, витраченого на процес шифрування. Вища часова вартість підвищує безпеку, вимагаючи більше часу для шифрування, але може призвести до зниження швидкості обробки.
- Вартість пам'яті: параметр, що регулює обсяг пам'яті, використаної для шифрування. Вища вартість пам'яті вимагає більше пам'яті для

шифрування, що збільшує обчислювальні витрати для зловмисника, який намагається зламати пароль.

- Ступінь паралелізму: параметр, що визначає кількість потоків або завдань, оброблюваних одночасно. Вищий ступінь паралелізму дозволяє проводити більше одночасних обробок, що призводить до швидшого шифрування.

Крім того, в алгоритмі Argon2 використовуються такі параметри, як кількість каналів, кількість ітерацій і тип. Ці параметри впливають на безпеку та продуктивність алгоритму Argon2 і потребують відповідного налаштування.

Внутрішньо алгоритм Argon2 використовує хеш-функцію Blake2. Він брав участь у конкурсі SHA-3, але в кінцевому підсумку програв Keccak. Blake2b є високопродуктивною криптографічною хеш-функцією, яка ефективно генерує фіксовані короткі дайджести для даних. Вона базується на оригінальному алгоритмі BLAKE та оптимізована для 64-бітних платформ [8]. Алгоритм використовує раундову функцію на основі блочного шифру для перетворення вхідних даних у хеш і включає побітові операції та нелінійні перетворення для зміни внутрішнього стану. Цей процес мінімізує ризики колізійних атак. Blake2b можна легко застосувати до різних випадків використання та вимог безпеки, використовуючи унікальні ключі, солі та параметри персоналізації. Загалом, Blake2b визнаний надійною криптографічною хеш-функцією, яка поєднує високу швидкість обробки з високою безпекою.

Крім того, Blake2b є алгоритмом, який дозволяє гнучко налаштовувати довжину вихідного хешу. Це означає, що довжина згенерованого хешу може бути налаштована відповідно до конкретних потреб. Як результат, Blake2b може створювати дайджести різної довжини, роблячи його універсальним для різних цілей. Ця гнучкість дозволяє широко використовувати Blake2b і полегшує генерацію оптимізованих дайджестів, пристосованих до конкретних застосувань. Таким чином, Blake2b не залежить від фіксованої довжини виходу і оцінюється як криптографічна хеш-функція, яка забезпечує як гнучкість, так і стабільність. В Argon2 хеш-функція Blake2 використовується у декількох ключових частинах:

– Ініціалізація: Blake2b використовується для генерації початкового блоку.

– Заповнення блоків: Blake2b використовується в обчисленні кожного блоку на основі попередніх блоків.

– Генерація фінального хешу: фінальний блок передається до хеш-функції Blake2b для генерації кінцевого хешу пароля.

В результаті всіх пройдених етапів ми отримуємо хеш-значення, як на рисунку 2.5.

Рисунок 2.5 – Хеш-значення отримане алгоритмом argon2

Підсумовуючи, Argon2 є потужним інструментом для захисту паролів та інших конфіденційних даних. Завдяки своїй адаптивності, використанню солі та високій складності обробки, Argon2 забезпечує ефективний захист від атак типу грубої сили та атак із використанням обчислювальних ресурсів. Це робить його ідеальним вибором для безпечного зберігання паролів у сучасних системах та додатках.

3 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ХЕШУВАННЯ

3.1 Характеристика пристрою та середовища використаних для дослідження

Для програмної реалізації алгоритмів та методів дослідження їхньої ефективності було обрано мову програмування Python та інтегроване середовище розробки PyCharm.

Python – це інтерпретована, динамічно типізована мова програмування високого рівня. Вона підтримує кілька парадигм програмування, включаючи об'єктно-орієнтоване, процедурне та функціональне, а також має автоматичне управління пам'яттю та велику стандартну бібліотеку, що робить її придатною для різноманітних завдань, від веб-розробки до наукових обчислень та аналізу даних.

Мова програмування Python має низку переваг, які роблять його ідеальним для проведення такого роду досліджень. По-перше, Python має багатий набір стандартних бібліотек і модулів, а також численні зовнішні бібліотеки, які підтримують різні алгоритми хешування, такі як bcrypt, argon2, scrypt та hashlib для PBKDF2. Це дозволяє легко імплементувати та тестувати різні алгоритми без потреби в розробці їх з нуля, що виключає помилки та непродуктивність коду. По-друге, існують потужні інструменти для профілювання та аналізу продуктивності, які дозволяють детально оцінювати ефективність алгоритмів. Крім цього, в даної мови програмування простий та зрозумілий синтаксис, Python дозволяє швидко писати, читати та розуміти код, що значно полегшує процес розробки та аналізу алгоритмів хешування.

Вибір метрик для оцінки ефективності алгоритмів хешування є критично важливим для отримання об'єктивних та репрезентативних результатів. У даній роботі було вирішено використовувати наступні метрики: час виконання, використання CPU, використання пам'яті.

Час виконання є однією з найважливіших метрик при оцінці алгоритмів хешування. Вона дозволяє виміряти, скільки часу потрібно алгоритму для обчислення хешу пароля. Алгоритми, які займають занадто багато часу для виконання, можуть значно знизити продуктивність системи, особливо якщо хешування відбувається часто. А також сучасні алгоритми хешування повинні бути досить повільними, щоб ускладнити атаки грубою силою. Однак, вони не повинні бути настільки повільними, щоб це негативно впливало на користувацький досвід.

Використання CPU є важливою метрикою, оскільки вона дозволяє оцінити, скільки обчислювальних ресурсів вимагає алгоритм хешування і чи не створює він надмірного навантаження на систему.

Використання пам'яті є критичною метрикою для оцінки ефективності алгоритмів хешування, оскільки воно впливає на масштабованість і загальну продуктивність системи. Деякі алгоритми можуть вимагати значних обсягів оперативної пам'яті, що може бути проблематичним для систем з обмеженими ресурсами.

Важливо вказати середовище в якому проводився аналіз алгоритмів хешування тому, що різні характеристики апаратного забезпечення можуть впливати на продуктивність алгоритмів хешування. Для порівняльної характеристики дослідження проводилися на двох операційних системах з різними потужностями. Характеристики основного, персонального комп'ютера мають наступні значення:

- Процесор: Intel Core i5-9300H, 4 ядра, 8 логічних процесорів, 2.4 GHz
- Оперативна пам'ять:16 GB DDR4
- Накопичувач: 500 GB NVMe SSD
- Операційна система: Windows 10 Pro 64-bit

Ефективність алгоритмів хешування може варіюватися в залежності від операційної системи, на якій вони виконуються. Запуск цих алгоритмів на Linux-системі може вплинути на їхню продуктивність через декілька факторів, а саме: керування пам’яттю, планування процесів, операції вводу та виводу, використання центрального процесора та систему файлів.

Тому для тестування використовувалася віртуальна машина на ядрі Linux, спеціально створена з обмеженою кількістю ресурсів. Для створення віртуальної машини було використано продукт призначений для віртуалізації, а саме VMware Workstation Pro 17 компанії VMware.

Отож, тестування проводилися на віртуальній (гостьовій) машині, що використовувала потужності основної машини з наступними характеристиками:

- Процесор: 1 ядро, 1 логічний процес
- Оперативна пам'ять: 2048Mb
- Накопичувач: 80 GB HDD
- Операційна система: Kali Linux

Характеристики, такі як процесор, оперативна пам'ять, тип і швидкість накопичувача, а також архітектура операційної системи можуть значно впливати на результати тестування алгоритмів хешування, вказавши їх можна адекватно інтерпретувати результати дослідження.

3.2 Розробка програм для дослідження обраних алгоритмів хешування

Як зазначалося вище, мова програмування Python була обрана для виконання тестів ефективності алгоритмів хешування завдяки своїй простоті, широкій підтримці криптографічних бібліотек та потужним інструментам для аналізу продуктивності. Основні бібліотеки, які використовуються для цієї роботи: hashlib, bcrypt, argon2, scrypt, timeit, psutil, memory_profiler, cProfile і pstats.

Hashlib – це стандартна бібліотека Python для роботи з різними алгоритмами хешування, що забезпечує простий інтерфейс для створення хешів з використанням алгоритмів SHA-2. Також була використані бібліотеки, які реалізують алгоритм хешування bcrypt та scrypt, також бібліотека argon2, відповідно для реалізації алгоритму хешування Argon2.

Бібліотека timeit – це стандартна бібліотека Python для вимірювання часу виконання невеликих фрагментів коду з високою точністю, що допомагає оцінити швидкодію алгоритмів хешування. Бібліотека psutil використовується

для моніторингу системних ресурсів, яка дозволяє отримувати інформацію про використання процесора, пам'яті та інші метрики. В межах дослідження вона виконує вимірювання використання CPU під час виконання функцій хешування. Бібліотека для профілювання використання пам'яті програмами на Python, що дозволяє оцінити обсяг оперативної пам'яті, необхідної для виконання алгоритмів хешування. Бібліотеки cProfile і pstats – це стандартні бібліотеки Python для профілювання та аналізу продуктивності коду. Вони використовуються для збору детальної інформації про час виконання функцій та ідентифікації потенційних вузьких місць у коді.

Принцип роботи програми можна розібрати на прикладі коду програми алгоритму хешування bcrypt. Код програми наведено в додатку А, а код самої функції хешування наведеного в лістингу 3.1.

Лістинг 3.1 – Код програми алгоритму хешування bcrypt

```
def hash_bcrypt():
    password = b"password"
    salt = bcrypt.gensalt()
    hashed = bcrypt.hashpw(password, salt)
    #print(f"Generated hash: {hashed.decode()}")
```

В програмі використовується декілька основних функцій, а саме: hash_bcrypt, profile_function та measure_cpu_usage. Функція hash_bcrypt призначена для хешування паролю. Вона бере початковий пароль, додає випадково згенеровану сіль для забезпечення унікальності хешу використовуючи алгоритм хешування bcrypt.Функція profile_function виконує профілювання переданої функції (bcrypt) за допомогою модуля cProfile наведена в лістингу 3.2.

Лістинг 3.2 – Код функції профілювання profile_function

```
def profile_function(func):
    pr = cProfile.Profile()
    pr.enable()
    func()
    pr.disable()
```

```
    s = StringIO()
    sortby = 'cumulative'
    ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
    ps.print_stats()
    print(s.getvalue())
```

Профілювання дозволяє отримати детальну статистику про те, скільки часу витрачається на виконання різних частин програми. Це важливо для виявлення вузьких місць у продуктивності коду. В даному випадку, результати сортуються за кумулятивним часом виконання (cumulative). Кумулятивний час – це сумарний час, витрачений на виконання певної функції, включаючи час, витрачений на всі функції, викликані з неї. В результаті можна отримати інформацію про кількість викликів функції, витрачений час на один виклик, час на виконання коду функції та кумулятивний час.

Наступна функція, що була використана для аналізу ефективності – це measure_cpu_usage, призначена для вимірювання використання процесора функцією, що тестується, за певну кількість ітерацій. Код даної функції наведений в лістингу 3.3.

Лістинг 3.3 – Код функції аналізу продуктивності measure_cpu_usage

```
def measure_cpu_usage(func, iterations=100):
    process = psutil.Process(os.getpid())
    start_cpu = process.cpu_times()
    start_time = time.time()

    for _ in range(iterations):
        func()
    end_time = time.time()
    end_cpu = process.cpu_times()
    user_time = end_cpu.user - start_cpu.user
    system_time = end_cpu.system - start_cpu.system
    elapsed_time = end_time - start_time
    cpu_usage = (user_time + system_time) / elapsed_time * 100
    cpu_count = psutil.cpu_count()
```

Функція надає детальну інформацію про час користувача, системний час, загальний час виконання та використання CPU. Ця функція є дуже важливою, адже допомагає зрозуміти, як ефективно функція використовує процесор, а

також може бути корисною для оптимізації продуктивності. Для більш чистого результату кількість ітерацій функції рівна ста. В результаті буде отримано користувацький час, системний час, загальний час виконання, загальне використання CPU у відсотках та використання CPU на одне ядро.

Для виміру кількості використаної пам’яті та часу роботи програми використовуються функції timeit.timeit та memory_usage відповідно. Код цих функцій наведений в лістингу 3.4.

Лістинг 3.4 – Код функції timeit.timeit та memory_usage

```
time_bcrypt = timeit.timeit(hash_bcrypt, number=1)
memory_bcrypt = memory_usage(hash_bcrypt, max_usage=True)
print(f"bcrypt: {time_bcrypt} seconds, {memory_bcrypt} MiB")
```

Крім вище наведених функцій є функції, що безпосередньо реалізують інші досліджуванні алгоритми хешування. Код функції, що реалізує алгоритм хешування scrypt наведено в лістингу 3.5.

Лістинг 3.5 – Код програми алгоритму хешування scrypt

```
def hash_scrypt():
    password = b"password"
    salt = os.urandom(16)
    hashed = hashlib.scrypt(password, salt, N=131072, r=8, p=1)
    #print(f"Generated hash (scrypt): {hashed.hex()}")
```

В функцію були передані параметри рекомендовані OWASP, а саме параметр мінімальної вартості процесора/пам’яті (n=131072), або ж (128 МіБ). Розмір блоку (r=8), має стале значення та рівний 1024 байтам. Та ступінь паралелізму (p=1), що визначає кількість незалежних потоків обчислень, що можуть виконуватися одночасно.

Ще однією функцією алгоритму хешування є argon2 з однойменної бібліотеки python. Код програми використаний для реалізації алгоритму хешування argon2 наведено в лістингу 3.6.

Лістинг 3.6 – Код програми алгоритму хешування argon2

```
def hash_argon2():
    password = b"password"
    ph = argon2.PasswordHasher()
    hashed = ph.hash(password)
    #print(f"Generated hash (argon2): {hashed}")
```

Функція одразу використовує сталі значення параметрів, таких як кількість ітерацій (часу), необхідних для обчислення хешу (t=3), мінімальний розмір пам'яті (m=65536), та ступінь паралелізму (p=4), можна змінювати параметри залежно від спроможності обчислювальної машини.

Частина коду програми, що реалізує алгоритм хешування PPBKDF2 наведено в лістингу 3.7.

Лістинг 3.7 – Код програми алгоритму хешування argon2

```
def hash_pbkdf2():
    password = b"password"
    salt = os.urandom(16)
    hashed = hashlib.pbkdf2_hmac('sha512', password, salt, 210000)
    #print(f"Generated hash (pbkdf2): {hashed.hex()}")
```

В функцію hash_pbkdf2 був переданий внутрішній алгоритм хешування SHA512, а також було встановлено значення кількості ітерацій (210000), що рекомендовано національним інститутом стандартів і технологій (NIST).

Отже, в процесі дослідження ефективності алгоритмів хешування було розроблено чотири програми на мові Python для безпосередньої оцінки алгоритмів хешування bcrypt, scrypt, argon2 та PPBKDF2. Було описано роботу програми та функцій, що виконують роботу збиранню інформації щодо ефективності алгоритмів, а саме за такими параметрами: використання центрального процесора, використання пам’яті, час роботи та кількість викликів функції.

3.3 Тестування та порівняння ефективності досліджуваних алгоритмів хешування

Для оцінки ефективності досліджуваних алгоритмів хешування, а саме bcrypt, scrypt, і Argon2, та PPBKDF2 було проведено серію експериментів. Ми вимірювали час виконання, споживання процесорного часу, використання пам’яті та CPU для кожного алгоритму за допомогою спеціально розробленої програми. Порівняльний аналіз є важливою частиною дослідження, оскільки він дозволяє розробникам і системним адміністраторам вибрати найкращий алгоритм хешування для хешування паролів користувачів. Наприклад, деякі алгоритми можуть забезпечувати вищий рівень безпеки за рахунок більшого використання ресурсів, тоді як інші можуть бути менш вимогливими до ресурсів, але забезпечувати нижчий рівень безпеки. Розуміння цих компромісів є ключовим для прийняття обґрунтованих рішень, що допоможуть забезпечити високий рівень захисту даних, мінімізуючи при цьому вплив на продуктивність системи. Отже, результати виконання програми на основній машині, що реалізує обрахунок ефективності алгоритм хешування bcrypt зображено на рисунку 3.1.

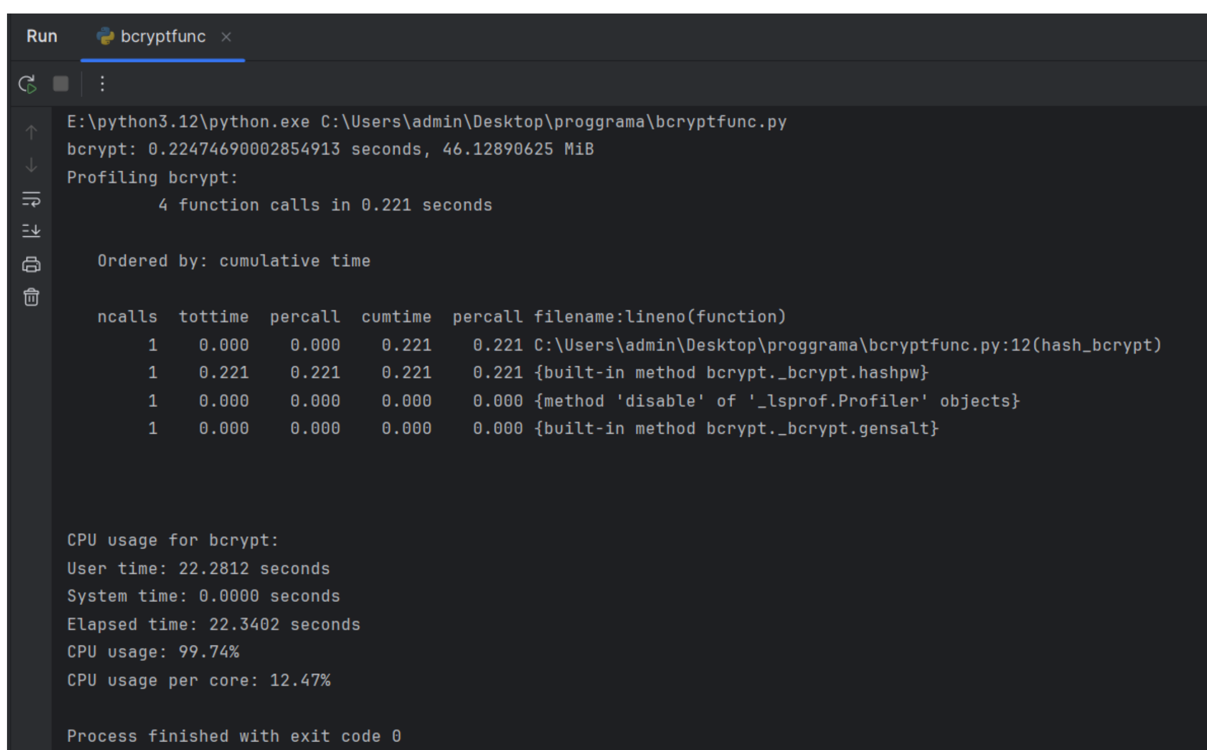

Рисунок 3.1 – Результат виконання програми алгоритму bcrypt

Функція хешування bcrypt демонструє відносно швидкий час виконання (0.2247 сек) та помірне використання пам'яті (46.1289 MB). Використання процесора на одне ядро складає 12.47%, що є досить низьким показником.

Результати виконання програми, що реалізує заміри ефективності алгоритму хешування scrypt зображено на рисунку 3.2.

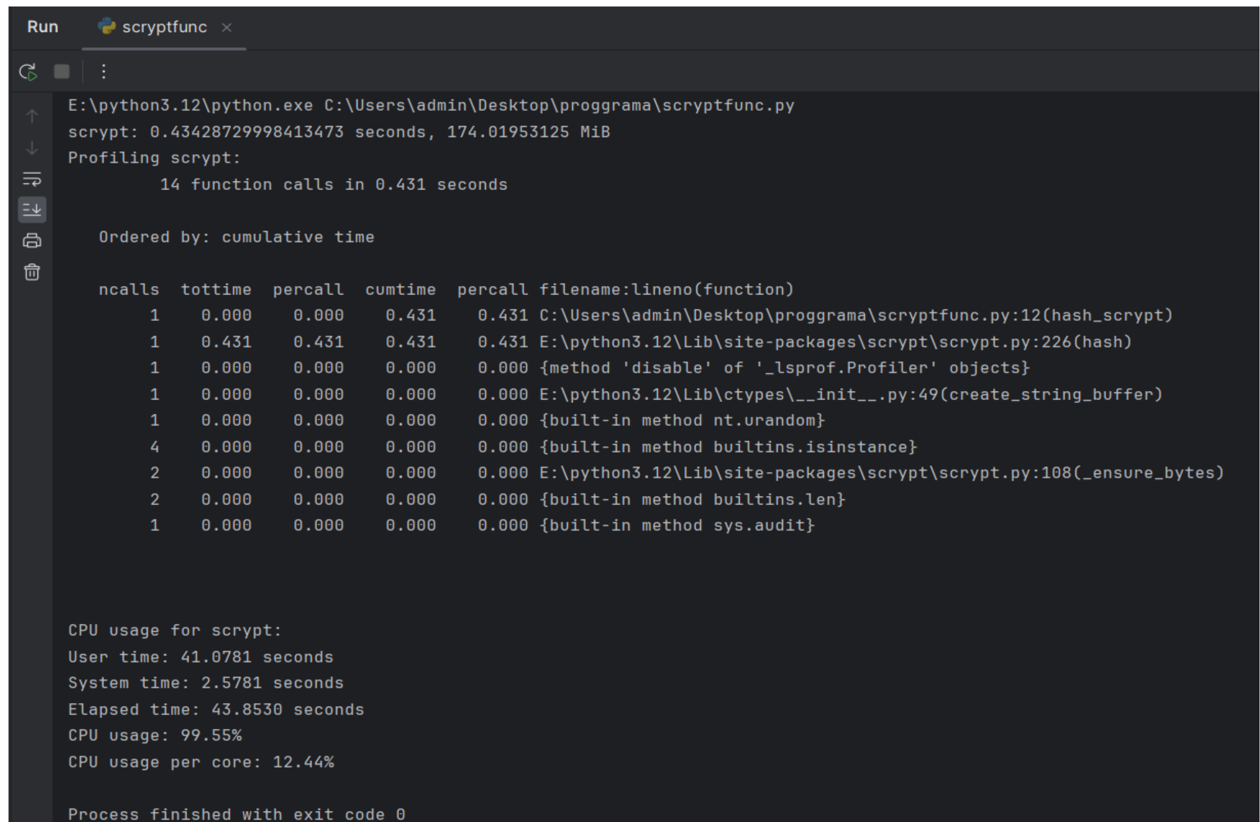

Рисунок 3.2 – Результат виконання програми алгоритму scrypt

Функція хешування scrypt має найдовший час виконання (0.4342 сек) і найвище використання пам'яті (174.0195 MB) серед усіх досліджуваних алгоритмів. Використання процесора на одне ядро складає 12.44%, що є схожим до показника bcrypt.

Результати виконання програми, що виконує оцінку ефективності алгоритму argon2 зображено на рисунку 3.3.

```
Run    argon2func ×

E:\python3.12\python.exe C:\Users\admin\Desktop\proggrama\argon2func.py
argon2: 0.07206989999394864 seconds, 110.15234375 MiB
Profiling argon2:
         38 function calls in 0.069 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.069    0.069 C:\Users\admin\Desktop\proggrama\argon2func.py:12(hash_argon2)
        1    0.000    0.000    0.069    0.069 E:\python3.12\Lib\site-packages\argon2\_password_hasher.py:148(hash)
        1    0.000    0.000    0.069    0.069 E:\python3.12\Lib\site-packages\argon2\low_level.py:53(hash_secret)
        1    0.069    0.069    0.069    0.069 {built-in method _ffi.argon2_hash}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        1    0.000    0.000    0.000    0.000 E:\python3.12\Lib\site-packages\argon2\_password_hasher.py:144(type)
        1    0.000    0.000    0.000    0.000 E:\python3.12\Lib\site-packages\argon2\_password_hasher.py:124(time_cost)
        1    0.000    0.000    0.000    0.000 E:\python3.12\Lib\site-packages\argon2\_password_hasher.py:136(hash_len)
        1    0.000    0.000    0.000    0.000 E:\python3.12\Lib\site-packages\argon2\_password_hasher.py:140(salt_len)



CPU usage for argon2:
User time: 23.7188 seconds
System time: 2.5781 seconds
Elapsed time: 8.3713 seconds
CPU usage: 314.13%
CPU usage per core: 39.27%
```

Рисунок 3.3 – Результат виконання програми алгоритму argon2

Функція Argon2 демонструє дуже швидший час виконання (0.0720 сек.) при високому використанні пам'яті (110.1523 MB). Використання процесора на одне ядро складає 39.27%.

Результати виконання програми, використаної для обрахунку ефективності алгоритму хешування PPBKDF2 зображено на рисунку 3.4.

```
Run    pbkdf2func ×

E:\python3.12\python.exe C:\Users\admin\Desktop\proggrama\pbkdf2func.py
pbkdf2: 0.20856780000030994 seconds, 45.6015625 MiB
Profiling pbkdf2:
         4 function calls in 0.204 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.204    0.204 C:\Users\admin\Desktop\proggrama\pbkdf2func.py:12(hash_pbkdf2)
        1    0.204    0.204    0.204    0.204 {built-in method _hashlib.pbkdf2_hmac}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        1    0.000    0.000    0.000    0.000 {built-in method nt.urandom}



CPU usage for pbkdf2:
User time: 20.0781 seconds
System time: 0.0000 seconds
Elapsed time: 20.1106 seconds
CPU usage: 99.84%
CPU usage per core: 12.48%

Process finished with exit code 0
```

Рисунок 3.4 – Результат виконання програми алгоритму PPBKDF2

Функція PBKDF2 має час виконання (0.2085 сек), що є подібним до bcrypt, і використання пам'яті (45.6015 MB). Використання процесора на одне ядро складає 12.48%, що також є подібним до bcrypt та scrypt.

На основі отриманих результатів тестування на основній машині під управлінням операційної системи Windows 10 можна створити таблицю з порівняльними характеристиками, яка допоможе провести адекватну оцінку та виявити сильні та слабкі сторони кожного з досліджуваних алгоритмів. Отримані результати показників ефективності досліджуваних алгоритмів хешування наведені в таблиці 3.1.

Таблиця 3.1 – Результати показників ефективності алгоритмів хешування

Функція хешування	Час роботи (сек.)	Вико-ристання пам'яті (MB)	Виклики функції	Вико-ристання процесора користувачем/ системою (сек.)	Вико-ристання процесора (%)	Вико-ристання процесора на одне ядро (%)
bcrypt	0.2247	46.1289	4	22.2812/0.0590	99.74	12.47
scrypt	0.4342	174.0195	14	41.0781/2.5781	99.55	12.44
argon2	0.0720	110.1523	38	23.7188/2.5781	314.13	39.27
PPBKDF2	0.2085	45.6015	4	20.0781/0,0325	99.84	12.48

Тепер повторимо тестування, але на гостьовій віртуальній машині під управлінням операційної системи Kali Linux, та обмеженими ресурсами. Отримані результати показників ефективності алгоритмів хешування наведені в таблиці 3.2.

Таблиця 3.2 – Результати показників ефективності алгоритмів хешування

Функція хешування	Час роботи (сек.)	Вико-ристання пам'яті (MB)	Виклики функції	Вико-ристання процесора користувачем/ системою (сек.)	Вико-ристання процесора (%)
bcrypt	0.2086	47.8632	17	20.8900/0.2561	98.79
scrypt	0.3472	176.5273	14	31.5800/2.1200	97.82
argon2	0.1541	112.5664	38	15.7117/1.0700	97.12
PPBKDF2	0.1581	47.7734	4	15.9100/0,0200	96.56

Порівняно з тестуванням на основній машині під управлінням Windows 10, результати на віртуальній машині під управлінням Kali Linux показують деякі відмінності у продуктивності алгоритмів. Bcrypt та scrypt показали менший час роботи. Argon2 продемонстрував більший час роботи, через наявність лише одного ядра та одного логічного процесу, але все ще залишився дуже ефективним. Алгоритм PBKDF2 залишився стабільним і продемонстрував менший час роботи.

Ці результати свідчать про важливість врахування середовища виконання при виборі алгоритму хешування для конкретного застосування. Різні операційні системи та конфігурації ресурсів можуть впливати на ефективність алгоритмів хешування, що повинно враховуватися під час їх впровадження.

Bcrypt показав стабільні результати на обох платформах. Він продемонстрував хороший баланс між часом виконання та використанням ресурсів. Bcrypt є відмінним вибором для загальних випадків, де важливий баланс між безпекою та використанням ресурсів.

Scrypt відзначився високим використанням пам'яті, що робить його стійким до атак на основі апаратних засобів, таких як GPU. На обох платформах він показав стабільний час роботи, але використання пам'яті залишилось високим. Це робить scrypt менш придатним для середовищ з обмеженою пам'яттю, але дуже ефективним для додаткового захисту у випадках, коли пам'ять не є критичним ресурсом.

Argon2 показав найшвидший час роботи серед всіх тестованих алгоритмів. Це робить argon2 відмінним вибором для сценаріїв, де важлива швидкість виконання, але також є достатньо ресурсів пам'яті та процесора. Його високий рівень використання процесора робить його ефективним для захисту від атак на основі апаратних засобів, але може бути обмеженням для менш потужних систем.

Алгоритм PBKDF2 продемонстрував стабільну продуктивність з помірним використанням ресурсів на обох платформах. Його час роботи та використання пам'яті залишилися майже незмінними, що свідчить про його надійність і стабільність. Цей алгоритм є хорошим вибором для загальних випадків використання, де необхідна стабільна продуктивність без високих вимог до ресурсів.

Отже, порівняльний аналіз ефективності алгоритмів хешування показує, що кожен з них має свої унікальні характеристики і підходить для різних сценаріїв використання. Тому вибір алгоритму хешування повинен враховувати специфічні вимоги середовища виконання та ресурси системи.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Працездатність людини-оператора

Людина-оператор – це людина, яка виконує роботу, що базується на взаємодії з предметом праці, машиною та зовнішнім середовищем через інформаційні моделі та органи керування [9].

Основною формою діяльністю людини-оператора є використання та опрацювання інформації. Це означає, що діяльність оператора складається із прийняття інформації, оцінювання та переробка інформації, прийняття рішення і в кінцевому результаті реалізація прийнятого рішення.

У процесі роботи людина переживає низку функціональних станів, які визначають різні рівні її працездатності. Під працездатністю людини розуміють можливість її виконувати певну роботу з необхідною якістю та у встановлений час. Відповідно зовнішні та внутрішні фактори в певній мірі впливають на продуктивність людини. Зовнішні фактори включають кількість та форму отриманої інформації, зручність робочого місця, тип взаємовідносин у колективі та вплив факторів навколишнього середовища. Внутрішні фактори включають рівень підготовки, тренованість і емоційну стійкість людини.

Розглядаючи зміни функціонального стану та якості роботи людини у процесі одного трудового циклу, виділяють 4 фази працездатності: пристосування до праці. стійкої працездатності, субкомпенсації, втоми [9].

Фаза пристосування до праці – це час, протягом якого людина адаптується до майбутніх умов праці. Поступово основний показник досягає свого встановленого значення. Тривалість пристосування організму до умов праці залежить від багатьох факторів, але основними з них є інтенсивність роботи (чим інтенсивніша робота, тим коротший період) та рівень готовності людини до майбутньої роботи. Підготовка людини до роботи (виконання фізичних вправ, адаптація зору та слуху та ін.) і посилене навчальне навантаження можуть значно скоротити фазу пристосування до праці.

У фазі стійкої працездатності відзначається найвища якість праці при найкращих рівнях функціонування фізіологічних систем організму. Ця тривалість залежить від кількості роботи. Цей проміжок часу зменшується з інтенсивністю роботи. Найкраща динамічна робота має тривалість, яка може бути в десятки разів більшою, ніж у статичній роботі. Емоції мають значний вплив на процес стійкої працездатності. Неприємні почуття знижують продуктивність, а позитивні якості, такі як впевненість, спокій і позитивний настрій, значно підвищують продуктивність. Продовження періоду стійкої працездатності можна забезпечити наступними чинниками:

- оптимальним рівнем напруги психофізіологічних функцій;
- комфортними умовами праці;
- правильним поєднанням роботи та відпочинку;
- емоційним розвантаженням;
- використанням тонізуючих напоїв (кава, чай), фармакологічних засобів, а саме препаратів рослинного походження (вітаміни, препарати, які впливають на енергетичні та метаболічні процеси);
- інформуванням людини про наслідки її діяльності, наглядом та контролем її роботи [9].

Практичний досвід показує, що вживання легких стимуляторів зменшує сонливість і підвищує працездатність на короткий період. Тим не менш, використання активних стимуляторів під час важкої роботи може призвести до погіршення самопочуття, зниження рухливості та швидкості реакцій. Вживання транквілізаторів заспокоїть та зможе запобігти неврозам. Однак вони також можуть знизити психічну активність, сповільнити реакції, спричинити апатію та сонливість.

Фаза субкомпенсації розглядається як початок розвитку втоми. У цей період якість праці все ще висока, але лише завдяки перенапруженню відповідних систем організму.

У фазі втоми спостерігається явне зниження якості роботи, а потім зниження функціонального стану людини. Зміна частоти пульсу, дихання, зорової та слухової чутливості є очевидними ознаками втоми.

Відновлення працездатності, або фаза відпочинку, повинна бути наступною фазою життєздатності людини. Ця фаза може тривати від декількох хвилин до декількох годин або навіть декількох діб.

Отже, забезпечення оптимальних умов праці для операторів, як і врахування фізіологічних та психологічних аспектів їх працездатності, є критично важливими для забезпечення ефективності та безпеки робочого процесу. Такий підхід до організації праці сприяє створенню ефективних та безпечних умов праці, що є важливим для успішного функціонування будь-яких систем.

4.2 Вимоги до режимів праці і відпочинку при роботі з ВДТ

Для підтримки працездатності та збереження здоров’я працівників під час організації праці з використанням ВДТ (візуальними дисплейними терміналами), ЕОМ (електронно-обчислювальних машинах) та ПЕОМ (персональних ЕОМ) передбачаються регламентовані перерви для відпочинку в межах зміни. Внутрішньозмінні режими праці та відпочинку включають додаткові короткі перерви в періоди, що передують симптомам стомлення та зниження працездатності.

При виконанні робіт, що належать до різних видів трудової діяльності, за основну роботу з ВДТ слід вважати таку, що займає не менше 50% робочого часу. Впродовж робочої зміни мають бути передбачені:

- проміжки часу, призначені для відпочинку та вживання їжі (обідні перерви);
- додаткові час для відпочинку та особистих потреб (згідно з трудовими нормами);
- додаткові перерви для певних професій відповідно до особливостей трудової діяльності [10].

Згідно з діючим класифікатором професій, розрізняють три професійні групи за характером роботи, а саме:

- розробники програм (інженери-програмісти) здебільшого працюють з відеотерміналами та документацією, оскільки потрібен інтенсивний обмін інформацією з електронними комп'ютерами та висока частота прийняття рішень. Робота характеризується інтенсивною розумовою та творчою працею, підвищеним напруженням зору, концентрацією уваги, вимушеною робочою позою, загальною гіподинамією та періодичним навантаженням на кисті верхніх кінцівок. Робота виконується в режимі діалогу з ЕОМ у вільному темпі з періодичним пошуком помилок в умовах дефіциту часу;
- оператори електронно-обчислювальних машин виконують роботу, пов'язану з обліком інформації, отриманої з ВДТ за попереднім запитом, або тієї, що надходить з нього, супроводжується перервами різної тривалості, пов'язана з виконанням іншої роботи, і характеризується напруженням зору, невеликими фізичними зусиллями, нервовим напруженням середнього ступеня та виконується у вільному темпі;
- оператор комп'ютерного набору виконує одноманітну роботу з документацією та клавіатурою з рідкісними короткими переведенням погляду на дисплей і введенням даних з високою швидкістю. Робота характеризується як фізична праця з підвищеним навантаженням на кисті верхніх кінцівок. Це супроводжується загальною гіподинамією, напруженням зору (концентрація на документах), нервово-емоційним навантаженням [10].

Правилами встановлюються наступні внутрішньозмінні режими праці та відпочинку при роботі з ЕОМ при 8-годинній денній робочій зміні в залежності від характеру праці:

- через кожну годину роботи за ВДТ розробникам програм із застосуванням ЕОМ слід призначати регулярну перерву для відпочинку тривалістю 15 хвилин;
- через кожні дві години операторам ЕОМ слід призначати регулярні перерви для відпочинку тривалістю 15 хвилин;

– після кожної години роботи за ВДТ операторам комп’ютерного набору слід призначати регламентовані перерви для відпочинку тривалістю 10 хвилин [10].

У будь-якому випадку, якщо виробничі обставини не дозволяють регламентованих перерв, безперервна робота з ВДТ не повинна перевищувати чотири години. При 12-годинній робочій зміні регламентовані перерви повинні встановлюватися в перші 8 годин роботи аналогічно перервам при 8-годинній робочій зміні, а протягом останніх 4-х годин роботи, незалежно від характеру трудової діяльності, через кожну годину тривалістю 15 хвилин.

Отже, ефективне використання регламентованих перерв та внутрішньозмінних режимів праці є важливою складовою для забезпечення здоров'я та підтримки працездатності працівників, які працюють з ВДТ. Перерви дозволяють зменшити негативні наслідки тривалого перебування перед екраном, підвищуючи концентрацію та знижуючи фізичне та психічне напруження.

4.3 Системи протипожежної безпеки в ЦОД

Останнім часом значно зросла кількість сховищ інформації – приміщень з комп'ютерними серверами, масивами жорстких дисків, мережевих комутаторів, пристроїв розподілу інтернет-трафіку. Їх функціонування передбачає забезпечення безперервної роботи з підтриманням постійного температурного режиму та забезпеченням фізичного збереження апаратури. Пожежі можуть призвести до непоправних втрат інформації. Тому при організації пожежної безпеки в центрах опрацювання даних (ЦОД) на перший план виходить вирішення завдання раннього виявлення загорянь.

Розрізняють декілька сценаріїв розвитку пожежі в центрах обробки даних. Перший сценарій передбачає тління проводів або мікросхем. Шкода заподіюється газом, що виділяється при тлінні та горінні, який окисляє електронні контакти. Рішенням проблеми є використання газоаналізаторів, вбудованих в систему раннього виявлення пожежі. У випадках сильного задимлення простору в роботу вступають неадресні датчики диму, встановлені

всередині стійок. Відправляються ними сигнали обробляє програма моніторингу серверного обладнання, встановленого в дата-центрі. Ці датчики використовують як джерело додаткової інформації для контролю всіх систем даного об'єкту а також для активації системи пожежогасіння окремої стійки [11].

Наступний сценарій полягає в сильному задимленні приміщення ЦОД. Виявляється завдяки адресним або неадресним датчикам загальної системи пожежної сигналізації, встановлених всередині приміщення дата-центру. Пожежні панелі, що виступають в якості центрального приймально-контрольного устаткування, при виявленні загоряння автоматично запускають системи активного пожежогасіння.

При оснащенні дата-центрів вибирається один з наступних методів ліквідації загорянь:

- Гіпоксичний метод – створення і підтримання атмосфери, в якій пожежа не може виникнути. У приміщення вводиться азот, який зменшує вміст кисню до мінімального рівня. У такій атмосфері вогонь не може виникнути і поширюватися, і рівень кисню тримається достатнім для роботи в серверному приміщенні.
- Ізоляція – працює за принципом заміщення кисню. У приміщення з пожежею подається чистий інертний газ або суміш для пожежогасіння. Рівень кисню зменшується і полум'я гасне. Використовується азот, аргон, аргоніт або інерген.
- Інгібування – передбачає впорскування галогенізованого газу, який гальмує хімічні реакції в полум'ї, тим самим перешкоджаючи процесу горіння.
- Порошок або аерозоль – відбувається викид порошкової хімії і розпорошення аерозолю. І порошок, і аерозоль на поверхні палаючих предметів утворюють плівку, що запобігає проникнення кисню, що знижує ймовірність повторного загоряння [11].

Найбільш популярним на сьогодні є метод газового пожежогасіння. Газ не шкодить електрообладнанню і добре працює навіть у важкодоступних приміщеннях.

ВИСНОВКИ

У роботі було описано проблему зберігання паролів користувачів, а також проведено детальний аналіз криптографічних властивостей, вимірювання ефективності та порівняння чотирьох поширених алгоритмів хешування паролів користувачів: bcrypt, scrypt, Argon2 і PBKDF2. Метою дослідження було проаналізувати та оцінити ефективність цих алгоритмів та придатність для практичного застосування. Для цього було розроблено програму, яка виконувала заміри ефективності досліджуваних алгоритмів за різними параметрами, включаючи час виконання, використання пам'яті, споживання процесорного часу та загальну продуктивність. Результати цього дослідження надають цінну інформацію для вибору оптимального алгоритму хешування в залежності від конкретних вимог і обмежень системи.

Ці результати мають велике значення для розробників та системних адміністраторів, які обирають алгоритм хешування для своїх застосувань. Знання характеристик кожного алгоритму дозволяє приймати обґрунтовані рішення, забезпечуючи баланс між безпекою, продуктивністю та використанням ресурсів.

Опираючись на дослідження, можна сказати, що bcrypt є надійним вибором для більшості застосувань завдяки своїй збалансованій продуктивності, scrypt рекомендується для середовищ, де важлива стійкість до апаратних атак, але наявні достатні ресурси пам'яті. Натомість argon2 є оптимальним для високопродуктивних систем, де важлива швидкість і використання багатоядерності, а PBKDF2 забезпечує стабільну продуктивність і підходить для загальних випадків використання.

Вибір алгоритму хешування повинен враховувати конкретні вимоги до системи та ресурсів, що дозволяє забезпечити максимальну безпеку та ефективність. Ця робота надає чіткі рекомендації та порівняльний аналіз, що може бути корисним для широкого кола застосувань у галузі кібербезпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Authentication and Authorization in Modern Web Apps for Data Security Using Nodejs and Role of Dark Web / P. Pant та ін. Procedia Computer Science. 2022. С. 781–790. URL: https://www.sciencedirect.com/science/article/pii/S1877050922021512#abs0001 (дата звернення: 11.04.2024).

2. KOCATEKİN T. Parola Saklama Tekniklerinin Evrimi ve Güncel En İyi Uygulamaları. Eskişehir Türk Dünyası Uygulama ve Araştırma Merkezi Bilişim Dergisi. 2023. URL: https://doi.org/10.53608/estudambilisim.1318760 (дата звернення: 17.05.2024).

3. Password Storage Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 11.04.2024).

4. Букатка, С., & Тимощук, В. (2023). ХЕШ-алгоритм шифрування паролів користувачів ос Linux. Матеріали VI Міжнародної студентської науково-технічної конференції „Природничі та гуманітарні науки. Актуальні питання “, 112-113.

5. Iuorio A. F., Visconti A. Understanding Optimizations and Measuring Performances of PBKDF2. 2nd International Conference on Wireless Intelligent and Distributed Environment for Communication, м. Milan, 11 лют. 2019 р. 2019. С. 101–114. URL: https://eprint.iacr.org/2019/161.pdf (дата звернення: 10.05.2024).

6. Стебельський, М., & Букатка, С. (2023). Загальносистемні криптографічні політики ОС Linux. Порівняльний аналіз. Матеріали VI Міжнародної студентської науково-технічної конференції „Природничі та гуманітарні науки. Актуальні питання “, 177-178.

7. Optimized Implementation of Argon2 Utilizing the Graphics Processing Unit / S. Eum та ін. Applied Sciences. 2023. Т. 13, № 16. С. 9295. URL: https://doi.org/10.3390/app13169295 (дата звернення: 23.05.2024).

8. Karnaukhov, A., Tymoshchuk, V., Orlovska, A., & Tymoshchuk, D. (2024). USE OF AUTHENTICATED AES-GCM ENCRYPTION IN VPN. Матеріали конференцій МЦНД, (14.06. 2024; Суми Україна), 191-193. https://doi.org/10.62731/mcnd-14.06.2024.004

9. Revniuk O.A., Zagorodna N.V., Kozak R.O., Karpinski M.P., Flud L.O. "The improvement of web-application SDL process to prevent Insecure Design vulnerabilities". Applied Aspects of Information Technology. 2024; Vol. 7, No. 2: 162–174. DOI:https://doi.org/10.15276/aait.07.2024.12.

10. Журавель М. О. ТЕКСТИ (КОНСПЕКТ) ЛЕКЦІЙ з дисципліни «Цивільний захист і охорона праці в галузі». URL: https://zp.edu.ua/sites/default/files/konf/konspekt_lekciy_opg_2020_menedzhment.pdf (дата звернення: 27.05.2024).

11. Skorenkyy, Y., Kozak, R., Zagorodna, N., Kramar, O., & Baran, I. (2021, March). Use of augmented reality-enabled prototyping of cyber-physical systems for improving cyber-security education. In Journal of Physics: Conference Series (Vol. 1840, No. 1, p. 012026). IOP Publishing.

12. Top 200 Most Common Passwords. NordPass. URL: https://nordpass.com/most-common-passwords-list/ (дата звернення: 20.04.2024).

13. Secure Website Authentication using Hashing Algorithms / S. Sharma та ін. International Journal for Scientific Research & Development. 2019. С. 528–530.

14. Choi H., Seo S. C. Optimization of PBKDF2 Using HMAC-SHA2 and HMAC-LSH Families in CPU Environment. IEEE Access. 2021. Т. 9. С. 40165–40177. URL: https://doi.org/10.1109/access.2021.3065082 (дата звернення: 16.05.2024).

15. Batubara T. P., Efendi S., Nababan E. B. Analysis Performance BCRYPT Algorithm to Improve Password Security from Brute Force. Journal of Physics: Conference Series. 2021. Т. 1811, № 1. С. 012129. URL: https://doi.org/10.1088/1742-6596/1811/1/012129 (дата звернення: 20.05.2024).

ДОДАТОК

Додаток А Код програми, що реалізує заміри ефективності алгоритмів хешування

```
import os
import timeit
import bcrypt
import psutil
import time
from memory_profiler import memory_usage
import cProfile
import pstats
from io import StringIO


def hash_bcrypt():
    password = b"password"
    salt = bcrypt.gensalt()
    hashed = bcrypt.hashpw(password, salt)
    #print(f"Generated hash: {hashed.decode()}")


def profile_function(func):
    pr = cProfile.Profile()
    pr.enable()
    func()
    pr.disable()
    s = StringIO()
    sortby = 'cumulative'
    ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
    ps.print_stats()
    print(s.getvalue())


def measure_cpu_usage(func, iterations=100):
    process = psutil.Process(os.getpid())
    start_cpu = process.cpu_times()
    start_time = time.time()

    for _ in range(iterations):
        func()

    end_time = time.time()
    end_cpu = process.cpu_times()

    user_time = end_cpu.user - start_cpu.user
    system_time = end_cpu.system - start_cpu.system
    elapsed_time = end_time - start_time

    cpu_usage = (user_time + system_time) / elapsed_time * 100
```

```
    cpu_count = psutil.cpu_count()

    print(f"User time: {user_time:.4f} seconds")
    print(f"System time: {system_time:.4f} seconds")
    print(f"Elapsed time: {elapsed_time:.4f} seconds")
    print(f"CPU usage: {cpu_usage:.2f}%")
    print(f"CPU usage per core: {cpu_usage / cpu_count:.2f}%")


if __name__ == '__main__':
    time_bcrypt = timeit.timeit(hash_bcrypt, number=1)
    memory_bcrypt = memory_usage(hash_bcrypt, max_usage=True)
    print(f"bcrypt: {time_bcrypt} seconds, {memory_bcrypt} MiB")

    print("Profiling bcrypt:")
    profile_function(hash_bcrypt)
    print("CPU usage for bcrypt:")
    measure_cpu_usage(hash_bcrypt)
```