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

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: Методи та засоби підвищення ефективності безпеки комп'ютеризованої системи за допомогою виявлення вразливих хеш-стрічок

Виконав: студент(ка) 6 курсу, групи СІм-62
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

(підпис) Цвірла О.В. (прізвище та ініціали)

Керівник (підпис) Луцик Н.С. (прізвище та ініціали)

Нормоконтроль (підпис) Тиш Є.В. (прізвище та ініціали)

Завідувач кафедри (підпис) Осухівська Г.М. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2025

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

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

Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
«17» листопада 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Цвірла Олег Володимирович*
(прізвище, ім'я, по батькові)

1. Тема роботи *Методи та засоби підвищення ефективності безпеки комп'ютеризованої системи за допомогою виявлення вразливих хеш-стрічок*

Керівник роботи *Луцик Надія Степанівна*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *14* » *листопада 2025* року № 4/7-987
2. Термін подання студентом завершеної роботи *16 грудня 2025 р.*
3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1 Аналіз сучасних методів хешування та виявлення вразливих хеш-стрічок
2 Теоретична частина.
3 Практична частина.
4 Охорона праці та безпека в надзвичайних ситуаціях
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Актуальність і мета дослідження.
2. Завдання, об'єкт і предмет, наукова новизна і практична цінність дослідження.
3.
4.
5.
6.
7.
8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Осухівська Г.М., зав.каф. КС*	*04.12.2025*	
Безпека в надзвичайних ситуаціях	*Стручок В.С., ст. викладач каф. ОХ*	*05.12.2025*	

7. Дата видачі завдання *17.11.2025р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз сучасних методів хешування та виявлення вразливих хеш-стрічок.*	*17.11.2025р. – 23.11.2025р.*	*Викон.*
2.	*Теоретична частина.*	*24.11.2025р. – 02.12.2025р.*	*Викон.*
3.	*Практична частина.*	*03.12.2025р. – 10.12.2025р.*	*Викон.*
4.	*Охорона праці та безпека в надзвичайних ситуаціях*	*04.12.2025р. – 10.12.2025р.*	*Викон.*
5.	*Оформлення пояснювальної записки і графічного матеріалу*	*11.12.2025р. – 19.12.2025р.*	*Викон.*
6.	*Попередній захист кваліфікаційної роботи магістра*	*16.12.2025р.*	*Викон.*
7.	*Захист кваліфікаційної роботи магістра*	*23.12.2025р.*	

Студент _____ *Цвірла О.В.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Луцик Н.С.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Цвірла О.В. Методи та засоби підвищення ефективності безпеки комп'ютеризованої системи за допомогою виявлення вразливих хеш-стрічок: кваліфікаційна робота на здобуття кваліфікаційного ступеня магістра: спец. 123 — Комп'ютерна інженерія / наук. кер. Луцик Н.С. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2025.

Ключові слова: комп'ютеризована система, інформаційна безпека, хеш-функція, хеш-стрічка, машинне навчання, класифікація, Python, криптографія, захист даних, ентропійний аналіз, аудит безпеки.

Кваліфікаційна робота присвячена підвищенню рівня безпеки комп'ютеризованих систем шляхом автоматизованого виявлення криптографічно нестійких або аномальних хеш-стрічок, що використовуються для зберігання автентифікаційних даних, токенів доступу та контрольних сум в інформаційних сховищах. Проаналізовано властивості сучасних криптографічних хеш-функцій, типові вразливості та визначено ознаки, що дозволяють оцінювати рівень криптостійкості хеш-значень.

Описано математичну модель представлення хеш-стрічок та сформовано набір ознак, що включає структурні, символьні та ентропійні характеристики, а також визначені параметри, пов'язані з алгоритмом хешування.

Розроблено програмне забезпечення з модульною архітектурою, що включає компоненти криптографічного аналізу, ентропійного аналізу, підсистему машинного навчання, інструменти векторизації ознак та механізм класифікації.

Результати роботи можуть бути впроваджені у корпоративні системи аудиту інформаційної безпеки для проактивного контролю політик зберігання облікових даних, виявлення застарілих криптографічних алгоритмів та мінімізації ризиків несанкціонованого доступу до інформаційних ресурсів.

ANNOTATION

Tsvirla O.V. Methods and tools for improving the security efficiency of a computerized system by detecting vulnerable hash strings: Master's qualification thesis, specialty 123 — Computer Engineering / scientific advisor Lutsyk N.S. Ternopil: Ternopil Ivan Puluj National Technical University, 2025.

Keywords: computerized system, information security, hash function, vulnerable hash strings, machine learning, classification, Python, cryptography, data protection, entropy analysis, security audit.

This qualification thesis is devoted to enhancing the security of computerized systems through the automated detection of cryptographically weak or anomalous hash strings used for storing authentication data, access tokens, and checksums in information repositories. The study analyzes the properties of modern cryptographic hash functions, common vulnerabilities, and identifies key indicators that allow assessing the cryptographic strength of hash values.

A mathematical model for representing hash strings is presented, and a feature set is constructed that incorporates structural, symbolic, and entropy-related characteristics, along with parameters linked to the hashing algorithm.

Software with a modular architecture has been developed, incorporating components for cryptographic analysis, entropy evaluation, a machine learning subsystem, feature vectorization mechanisms, and a classification module.

The results of the work can be implemented in corporate information security audit systems to enable proactive enforcement of password storage policies, detection of outdated cryptographic algorithms, and minimization of risks related to unauthorized access to sensitive data

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) набір інструментів та правил для взаємодії між модулями програмного забезпечення.

ASIC (англ. Application-Specific Integrated Circuit) спеціалізована інтегральна схема, оптимізована для виконання криптографічних або перебірних операцій.

Argon2/PBKDF2/Bcrypt сучасні алгоритми хешування паролів, що забезпечують високу криптостійкість завдяки використанню параметрів часу та пам’яті.

Brute-force Attack метод повного перебору, спрямований на відновлення вихідного значення хешу.

Collision Attack криптографічна атака, що передбачає пошук двох різних повідомлень з однаковим хеш-значенням.

CPU (англ. Central Processing Unit) центральний процесор, універсальний обчислювальний елемент.

DBMS / СКБД (англ. Database Management System) система керування базами даних.

Feature Vector - вектор ознак числове представлення характеристик хешу, яке подається на вхід моделі машинного навчання.

GPU (англ. Graphics Processing Unit) графічний процесор, який застосовується для прискорених атак перебору хешів.

Hash Function (Хеш-функція) одностороннє криптографічне перетворення, що переводить дані у фіксовану довжину.

Hashcat/John the Ripper спеціалізовані інструменти для аудиту хешів та криптоаналізу

JSON (англ. JavaScript Object Notation) структуральний формат даних, що використовується для передачі результатів аналізу.

ML (англ. Machine Learning) методи машинного навчання, які використовуються для автоматизованої класифікації хешів.

MD5/SHA-1/SHA-2/SHA-3 криптографічні алгоритми хешування з різним рівнем криптостійкості.

SHA (англ. Secure Hash Algorithm) сімейство стандартів хешування.

Vulnerable Hash String (англ. Вразлива хеш-стрічка) хеш-значення, що має низьку криптостійкість або ознаки аномальності з точки зору структури, символів чи ентропії.

VHDS (англ. Vulnerable Hash Detection System) запропонована у роботі система виявлення вразливих хеш-стрічок.

Salt це випадкова, криптографічно якісна послідовність байтів, яку додають до пароля перед обчисленням хешу.

NIST (англ. National Institute of Standards and Technology) це національний інститут стандартів і технологій США, державна організація, що розробляє та публікує стандарти й рекомендації у сфері інформаційної безпеки, криптографії, цифрової ідентифікації та захисту даних.

OWASP (англ. Open Worldwide Application Security Project) це міжнародна некомерційна організація, що займається дослідженням, стандартизацією та поширенням найкращих практик у галузі безпеки веб-застосунків та інформаційних систем.

ASVS (англ. Application Security Verification Standard) це стандарт перевірки безпеки застосунків, розроблений OWASP, який визначає вимоги та рівні захищеності програмного забезпечення на різних етапах життєвого циклу.

HMAC (англ. Hash-based Message Authentication Code) це механізм автентифікації повідомлень на основі криптографічної хеш-функції та секретного ключа, що забезпечує цілісність і автентичність даних.

ECDSA (англ. Elliptic Curve Digital Signature Algorithm) це алгоритм цифрового підпису на основі еліптичних кривих, який використовується для перевірки автентичності та цілісності електронних даних.

ВСТУП

Актуальність роботи. Сучасні дослідження показують, що ігнорування вимог до вибору хеш-функції або помилки в її використанні призводять до практичних компрометацій реальних комп'ютерних систем. А також сучасні методи аудиту хеш-значень здебільшого ґрунтуються на перевірці відомих витоків, словникових атак або використанні статичних евристик. Такі підходи не дають можливості виявляти нові, приховані або структурно нетипові хеш-стрічки, що можуть бути ознакою використання слабких алгоритмів, пошкоджених даних чи потенційних загроз. Застосування криптоаналітичного аналізу, ентропійних характеристик та машинного навчання, дозволить здійснювати більш точну класифікацію, виявляти аномалії та підвищувати рівень автоматизації процесів контролю безпеки хешів без потреби у ручному втручанні. Таким чином, дослідження комбінованих методів виявлення вразливих хеш-стрічок є надзвичайно важливим як з наукової, так і з практичної точок зору.

Метою кваліфікаційної роботи є підвищення ефективності захисту комп'ютеризованої системи шляхом розроблення комбінованого методу автоматизованого виявлення вразливих хеш-стрічок на основі криптоаналітичного аналізу, ентропійних характеристик та машинного навчання.

Завдання кваліфікаційної роботи:

- провести аналітичний огляд сучасних алгоритмів хешування та типових вразливостей хеш-функцій;

- дослідити причини появи слабких, аномальних та криптографічно нестійких хеш-значень у комп'ютеризованих системах;

- проаналізувати існуючі засоби аудиту та інструменти криптоаналізу хеш-стрічок;

- описати математичну модель структури хеш-значень і сформувати набір ознак для їх класифікації;

- розробити комбінований метод виявлення вразливих хешів, що поєднує криптоаналітичний аналіз, ентропійні характеристики та машинне навчання;

- реалізувати програмний модуль у середовищі Python для автоматизованої класифікації хеш-стрічок;

- провести експериментальне дослідження ефективності розробленої системи та порівняти її з існуючими підходами.

Об'єкт дослідження: процеси забезпечення інформаційної безпеки комп'ютеризованих систем, що використовують криптографічні хеш-функції.

Предмет дослідження: методи та засоби автоматизованого виявлення вразливих хеш-стрічок на основі комбінованих підходів криптоаналізу та машинного навчання.

Методи дослідження: включають методи криптоаналізу хеш-функцій; статистичний та ентропійний аналіз символьних послідовностей; алгоритми машинного навчання (класифікація, виявлення аномалій); методи математичного моделювання; експериментальні дослідження продуктивності; методи порівняльного аналізу та системного підходу.

Наукова новизна дослідження полягає у розробленні комбінованого методу виявлення вразливих хеш-стрічок, що поєднує структурні, символьні, ентропійні та криптоаналітичні характеристики з інтелектуальними моделями машинного навчання. Такий підхід забезпечує можливість виявлення не лише відомих, а й нових типів слабких або аномальних хешів, які не можуть бути ідентифіковані традиційними засобами.

Практичне значення результатів кваліфікаційної роботи полягає у можливості використання розробленої системи в корпоративних службах аудиту безпеки та моніторингу, для виявлення застарілих або небезпечних хеш-алгоритмів, оцінювання стану захисту сховищ даних автентифікацій, а також для мінімізації ризику компрометації інформаційних ресурсів.

Публікації. Результати дослідження апробовано на XIV Міжнародній науково-практичній конференції молодих учених та студентів «Актуальні задачі сучасних

технологій», та XIII науково-технічній конференції «Інформаційні моделі, системи та технології», у вигляді тез конференцій.

Структура роботи. Робота складається з пояснювальної записки та графічної частини. Пояснювальна записка складається із вступу, 4 розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ МЕТОДІВ ХЕШУВАННЯ ТА ВИЯВЛЕННЯ ВРАЗЛИВИХ ХЕШ-СТРІЧОК

1.1. Роль хеш-функцій у забезпеченні інформаційної безпеки

Криптографічні хеш-функції сьогодні є одним із базових елементів практичної криптографії та інформаційної безпеки. Вони застосовуються скрізь, де потрібне надійне підтвердження цілісності даних, перевірка автентичності або побудова більш складних криптографічних механізмів, зокрема цифрових підписів, протоколів автентифікації, схем зберігання паролів, блокчейн-систем тощо [8, 9]. У переважній більшості сучасних протоколів розробник навіть не бачить хеш-функцію, вона захована всередині бібліотек та стандартів, але від її властивостей фактично залежить надійність усієї системи.

Формально криптографічну хеш-функцію можна розглядати як відображення яке показано у формулі (1.1):

$$h: \{0,1\}^* \rightarrow \{0,1\}^n, \tag{1.1}$$

де $\{0,1\}^*$- множина бітових послідовностей довільної довжини;

$\{0,1\}^n$- множина фіксованої довжини n біт.

Хеш-функція перетворює повідомлення довільного розміру (пароль, файл, блок журналу подій) у компактний «відбиток» фіксованої довжини [8]. До таких функцій висуваються три ключові вимоги: стійкість до відновлення прообразу (pre-image resistance), стійкість до знаходження другого прообразу (second pre-image resistance) та стійкість до колізій (collision resistance) [8, 10].

Необхідність використання хеш-функцій випливає з моделі загроз сучасних комп’ютеризованих систем. Навіть якщо зловмисник здобуде доступ до бази даних, де зберігаються хешовані паролі, ця інформація не повинна бути достатньою для прямого відновлення початкових паролів. Саме тому відповідні рекомендації NIST

наголошують на застосуванні стійких односторонніх функцій та спеціальних схем хешування паролів з використанням «солі» й параметрів складності [10-12].

На практиці роль хеш-функцій можна умовно поділити на кілька основних напрямів:

- забезпечення цілісності даних - під час зберігання або передавання файлів, транзакцій чи конфігураційних записів обчислюється контрольний хеш, який потім порівнюється з отриманим значенням. Будь-яке навіть незначне спотворення даних, із високою ймовірністю змінює хеш-значення, що дозволяє виявити факт модифікації [8].

- автентифікація користувачів - паролі користувачів не зберігаються у відкритому вигляді, натомість зберігається хеш із застосуванням спеціалізованих схем хешування паролів (Argon2, bcrypt, PBKDF2 тощо). Під час входу система повторно обчислює хеш від введеного пароля і порівнює з раніше збереженим значенням [9,11].

- побудова складніших криптографічних примітивів - хеш-функції лежать в основі алгоритмів цифрових підписів, протоколів узгодження ключів, схем аутентифікації повідомлень (HMAC) та інших механізмів, де важливо поєднати цілісність і захист від підробки [8].

- ідентифікація даних та адресація в сховищах - у системах контент-адресованого зберігання (наприклад, в окремих розподілених файлових системах або блокчейн-платформах) хеш-значення використовується як унікальний ідентифікатор блоку даних [8].

На рисунку 1.1 умовно показано місце хеш-функції в типових сценаріях захисту даних.

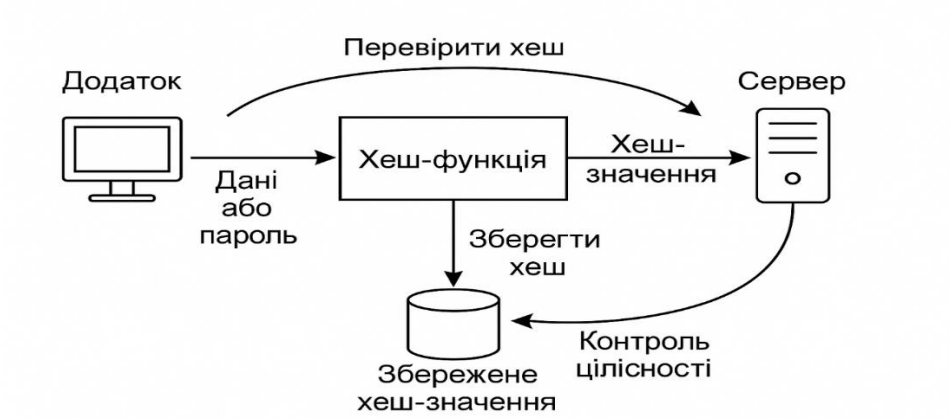

Рис 1.1. Хеш-функція в сценаріях захисту даних

Ключовою особливістю хеш-функцій є те, що безпека системи визначається не лише самим алгоритмом, а й контекстом його використання. Для зберігання паролів важливо застосовувати не просто «швидку» криптографічну хеш-функцію, а спеціалізовані ресурсомісткі схеми хешування паролів (password hashing schemes), які навмисно роблять кожну операцію обчислювально дорогою та часто потребують значного обсягу пам’яті [9,11]. Це необхідно для протидії атакувальникам, які застосовують багатопотокові та GPU/ASIC-рішення для масового перебору хешів [10].

Процес перевірки пароля складається з таких зміних P, S, f. Збереженя у базі значення пароля має вигляд як показано у формулі (1.2);

$$H = f(P, S, params), \tag{1.2}$$

де *params* - набір параметрів складності (кількість ітерацій, обсяг пам’яті, кількість потоків тощо);

P - пароль користувача;

S – випадкова «сіль»;

f - схема хешування паролів.

Під час автентифікації користувач вводить пароль P′, система обчислює як показано у формулі (1.3).

$$H' = f(P', S, params). \quad (1.3)$$

Після цього порівнює H' із збереженим H. Якщо значення збігаються, автентифікація вважається успішною. Важливо, що у цьому процесі ніколи не зберігається сам пароль P, а лише його хеш-представлення [10,12].

Сучасні дослідження показують, що ігнорування вимог до вибору хеш-функції або помилки в її використанні призводять до практичних компрометацій реальних систем. Роботи останніх років демонструють, що застосування швидких алгоритмів на кшталт MD5 або SHA-1 у ролі хешу паролів дає змогу зловмисникам із сучасним GPU-обладнанням проводити масові атаки перебором, обробляючи мільйони або навіть мільярди спроб за секунду [13,14]. Саме тому рекомендації OWASP та NIST однозначно вказують на необхідність використання спеціальних алгоритмів, таких як Argon2id, bcrypt або, у певних випадках, PBKDF2 із належно підібраними параметрами [11].

Таким чином, у контексті безпеки комп'ютеризованої системи хеш-функція виконує подвійну роль. З одного боку, вона є універсальним механізмом забезпечення цілісності й автентичності, без якого неможливо уявити сучасні криптографічні протоколи. З іншого боку, за умов неправильного вибору алгоритму або невдалих параметрів хешування саме хеш-представлення паролів перетворюється на «слабку ланку», яку першою атакують при витоку бази даних. Це й обґрунтовує необхідність подальшого аналізу властивостей хеш-функцій, вразливостей та методів виявлення небезпечних або аномальних хеш-стрічок у наступних підрозділах даного розділу.

Криптографічні хеш-функції становлять основу багатьох механізмів контролю цілісності та автентифікації, однак їхня надійність визначається стійкістю до трьох фундаментальних типів атак: атаки на відновлення прообразу**,** атаки на другий прообраз**,** та колізійні атаки**.** Порушення будь-якої з цих властивостей робить хеш-функцію непридатною для використання у сучасних системах захисту інформації. Актуальні криптографічні дослідження [15] підтверджують повну непридатність MD5 та SHA-1 для безпекових застосувань та вимагають переходу на SHA-2/SHA-3 або функції з пам'яттєвою складністю.

Атаки на відновлення прообразу (Pre-image attacks) - цей тип атаки передбачає спробу знайти будь-яке повідомлення x, яке після хешування дає заздалегідь відоме значення y. Якщо хеш-функція не забезпечує стійкість до такого відновлення, то зловмисник може підібрати повідомлення, що створює той самий хеш, і використати його для обходу механізмів автентифікації [16].

Формальне визначення хеш-функція описується:

$$h: \{0,1\}^* \rightarrow \{0,1\}^n, \tag{1.9}$$

де $h(\cdot)$ - хеш-функція;

$\{0,1\}^*$ - множина всіх двійкових рядків довільної довжини;

$\{0,1\}^n$ - множина двійкових рядків фіксованої довжини nnn біт.

Атака на відновлення прообразу — це пошук такого x:

$$h(x)=y, \tag{1.10}$$

де x - повідомлення, яке намагається знайти атакувальник;

y - хеш-значення, відоме зловмиснику.

Ймовірність випадкового знаходження прообразу обчислюється за формулою:

$$P = 2^{-n}, \tag{1.11}$$

де P - імовірність того, що випадкове повідомлення дасть потрібний хеш;

2^{-n}- зворотна величина простору можливих хешів.

Сучасні дослідження підтверджують, що для слабких функцій типу MD5 пошук прообразу значно прискорюється за рахунок паралельних GPU-обчислень [16].

Атаки на другий прообраз (Second pre-image attacks) - у цьому випадку зловмисник має конкретне повідомлення x і намагається підібрати інше повідомлення x', що формує ідентичний хеш, як показано у формулі (1.12):

$$h(x') = h(x), x' \neq x, \quad (1.12)$$

де x - початкове, відоме всім повідомлення;

x' - другий прообраз, який намагається знайти атакувальник;

$h(\cdot)$ - хеш-функція; умова x′≠x гарантує, що атака не є тривіальною.

Практичні дослідження [17] показують, що для SHA-1 існують оптимізовані варіанти second pre-image атак, які знижують складність у кілька разів, що підтверджує необхідність переходу на SHA-2/SHA-3.

Колізія - це ситуація, коли два різних повідомлення мають однакове хеш-значення як показано у формулі (1.13):

$$x_1 \neq x_2, h(x_1) = h(x_2). \quad (1.13)$$

Колізійні атаки використовують ефект дня народження, який зменшує складність пошуку збіжних значень.

Ймовірність знаходження колізії показано у формулі (1.14):

$$P_{collision} \approx 2^{-n/2}, \quad (1.14)$$

де $P_{collision}$- ймовірність знаходження колізії випадковим перебором;

n - довжина хешу;

$2^{-n/2}$- оцінка складності пошуку.

Сучасні зловмисники активно використовують обчислювальні платформи, які забезпечують мільярди хеш-операцій за секунду.

GPU і ASIC дозволяють реалізувати: повний перебір (brute-force), словникові атаки, комбіновані атаки (dictionary + rules), масивний паралелізм для pre-image і second pre-image атак.

Швидкість перебору NTLM або MD5 на високопродуктивному GPU сягає до 300–600 мільярдів хешів за секунду [20].

Це означає, що слабкі хеші фактично не забезпечують захисту, системи без «солі» стають компрометованими майже миттєво та безпека повністю залежить від алгоритму.

У відповідь на це було створено стійкі до GPU-атаки алгоритми (Argon2, bcrypt), які збільшують затрати пам’яті та час виконання.

1.2. Криптографічні властивості та класифікація сучасних хеш-алгоритмів

Сучасні хеш-функції походять із двох великих сімейств: алгоритми, побудовані на основі компресійних функцій за схемою Меркла-Дамґарда, та алгоритми, створені за конструкцією губки (sponge construction). Обидва підходи стали фундаментом для проєктування безпечних хеш-функцій, що застосовуються в цифрових підписах, аутентифікації, блокчейні та зберіганні паролів [21,22].

Загалом криптографічні хеш-функції повинні задовольняти три основні вимоги безпеки.

Перша вимога це стійкість до відновлення прообразу (pre-image resistance). Як показано у формулі (1.4) неможливо знайти будь-яке M, для якого виконується:

$$h(M)=H, \tag{1.4}$$

де H - відоме хеш-значення.

Друга вимога це стійкість до second pre-image атак. Як показано у формулі (1.5) неможливо підібрати інше повідомлення M′≠M, таке що:

$$h(M')=h(M). \tag{1.5}$$

Третя вимога це стійкість до колізій (collision resistance)**.** Як показано у формулі (1.6) практично неможливо знайти дві різні строки M_1 і M_2 , для яких хеш збігається:

$$h(M_1) = h(M_2). \tag{1.6}$$

У сучасних стандартах вважається, що для безпечної хеш-функції складність пошуку колізії має бути не меншою за $2^{n/2}$, де n - довжина вихідного хешу (відомий як принцип методу «дня народження») [23].

Алгоритми класичної криптографії. MD5 **і** SHA-1 історично відіграли ключову роль, однак сьогодні вважаються криптографічно небезпечними через практично реалізовані колізійні атаки. Починаючи з 2020 по 2024 роки, більшість стандартів рекомендують повну відмову від них [21,24].

Алгоритми SHA-2,SHA-256, SHA-512 основне сімейство, яке залишається безпечним при правильному використанні.

Алгоритми нового покоління - SHA-3 (на основі губкової конструкції Keccak) є альтернативою SHA-2, що забезпечує високу криптостійкість та унікальну структуру поглинання/вичавлення даних.

У SHA-3 функція h визначається як показано у формулі (1.7):

$$h(M) = Sponge[f, r, c](M), \tag{1.7}$$

де f - перестановка Keccak-f;

r - bitrate;

c - capacity.

Дані досліджень за 2021–2024 роки демонструють високу стійкість до сучасних криптоаналітичних атак [25].

Алгоритми хешування паролів (Password Hashing Schemes) відрізнюються від класичних хеш-функцій тим, що алгоритми не намагаються бути «швидкими», навпаки - їхня мета максимально ускладнити перебір.

До рекомендованих стандартами алгоритмів належать:

- Argon2id (переможець Password Hashing Competition; стійкість ґрунтується на комбінації споживання пам’яті, часу й паралельності) [26];

- bcrypt (досі актуальний, використовує адаптивну складність cost factor);

- PBKDF2 (стандартний, але потребує великої кількості ітерацій для захисту від GPU-атак).

Відповідно до OWASP ASVS 2023–2024, саме Argon2id рекомендовано як першочерговий вибір [27].

Формально процес виглядає так як:

$$H = Argon2id(P, S, t, m, p), \tag{1.8}$$

де P - пароль;

S - «сіль»;

t - кількість ітерацій;

m - обсяг пам’яті;

p - кількість потоків.

Для децентралізованих реєстрів і криптовалют використовуються функції зі спеціальними властивостями, серед яких: SHA-256d (Bitcoin); Keccak-256 (Ethereum); Blake2 / Blake3 - сучасні високопродуктивні алгоритми, що демонструють удвічі-тричі кращу продуктивність, ніж SHA-2, при збереженні криптостійкості [28].

Blake3 з 2020 по 2024 роки активно інтегрується у файлові системи та distributed storage, де потрібна велика продуктивність [29].

Хеш-функції в системах автентифікації та маркування даних.

У протоколах автентифікації (OAuth2, OpenID Connect, JWT) хеш-функції використовуються не лише для перевірки цілісності токенів, але й для формування унікальних підписів і claims-структур. Стандарти 2021–2024 років підкреслюють важливість переходу на SHA-256/384 у механізмах цифрових підписів HMAC та ECDSA [30].

1.3. Аналіз поширених випадків появи слабких та аномальних хеш-значень у комп’ютеризованих системах

У сучасних комп’ютеризованих системах слабкі або аномальні хеш-значення виникають значно частіше, ніж це здається під час теоретичного аналізу. Практика аудиту інформаційної безпеки, аналіз реальних витоків даних та результати

незалежних досліджень демонструють, що більшість вразливих хешів з’являються не через складні криптоаналітичні атаки, а через помилки розробників, неправильні налаштування або використання застарілих криптографічних механізмів [31].

На рисунку 1.2 наведено узагальнену класифікацію основних причин появи слабких та аномальних хешів у комп’ютеризованих системах, що охоплює криптографічні, конфігураційні та структурно-ентропійні фактори.

Рис 1.2. Узагальнена схема основних причин виникнення слабких та аномальних хеш-значень

Однією з найпоширеніших причин формування небезпечних хеш-значень є застосування алгоритмів, криптографічні властивості яких більше не відповідають сучасним вимогам. Типові представники:

-MD5 (повністю зламаний, колізії генеруються за секунди) [31];

-SHA-1 (уразливий до практичних колізій, заборонений для безпекових застосувань) [32];

-NTLM / LM Hash (відсутність «солі», робить перебір надзвичайно швидким на GPU).

У багатьох організаціях такі алгоритми використовуються внаслідок спадковості систем, недостатньої обізнаності розробників або через відсутність централізованої політики хешування. За дослідженнями NIST [33], понад 25% корпоративних систем з 2022 по 2023 роки містили щонайменше один тип застарілого хешування.

Сіль (salt) використовується для підвищення стійкості хеш-значень до масового перебору. Проте в реальних системах часто спостерігаються такі проблеми:

- повна відсутність «солі» (однакові паролі приводять до однакових хешів);
- статична «сіль» (одна змінна для всіх користувачів);
- коротка або низькоентропійна «сіль» (легко перебирається) [31];
- відсутність зберігання «солі» у базі або її некоректний формат.

Недостатня ентропія «солі» формує передумови для атак, у яких «сіль» перебирається швидше, ніж сам пароль. У багатьох реальних інцидентах (особливо у старих PHP/MySQL-системах) проблема виникала через реалізацію хешування без урахування сольових параметрів.

Аналіз великої кількості витоків даних свідчить, що частина хешів стає аномальною через технічні і логічні дефекти програмного забезпечення, зокрема:

- некоректне перетворення форматів (base64 ↔ hex);
- усічення хешів під час передачі API або при записі в БД;
- імпорт/експорт CSV з автоматичним обрізанням довгих значень;
- пошкодження частини хешу при серіалізації;
- змішування різних версій алгоритмів у одному полі БД.

Такі помилки часто призводять до появи хешів «неповної довжини» або таких, що не відповідають алфавіту обраного алгоритму.

Аномальні хеш-значення можуть виникати у випадках, коли системи: застосовують некриптографічні функції (CRC, Adler32), використовують «саморобні» алгоритми, генерують хеші з повторюваними патернами або мають вкрай низьку різноманітність символів.

Хеш із низькою ентропією практично завжди свідчить про небезпеку, навіть якщо формально він має правильну довжину. У роботі [34] зазначено, що близько 17% виявлених «хешів» у корпоративних системах виявлялися результатами простих XOR-маскувань або лінійних перетворень, які зовсім не є криптографічними.

Алгоритми на кшталт MD5, SHA-1 та навіть SHA-256 є надто швидкими для захисту паролів, оскільки їх можна обчислювати мільярдами за секунду на GPU. Це робить можливими атаки перебору навіть для складних паролів.

У звітах 2023–2024 років лабораторій GPU-обчислень [34] зазначається, що одна відеокарта NVIDIA RTX може розраховувати: до 600 млрд хешів MD5 за секунду, до 240 млрд хешів SHA-1 за секунду, понад 20 млрд SHA-256 за секунду.

Без уповільнюючих алгоритмів (bcrypt, Argon2, scrypt) такі хеші не забезпечують жодного реального захисту.

Одним із небезпечних явищ є використання так званих custom-hash алгоритмів - внутрішніх «розробок» компаній. Дослідження IEEE Cybersecurity Group [35] показують, що: понад 60% таких алгоритмів мають критичні криптографічні дефекти, більшість генерують передбачувані або низькоентропійні значення, їхню стійкість неможливо формально довести, атаки brute-force та reverse-engineering виявляються тривіальними.

Саме нестандартні алгоритми часто стають причиною появи хешів із аномальною структурою, про що свідчать численні аудити корпоративного ПЗ.

1.4. Огляд існуючих інструментів і сервісів аудиту хешів

У сучасних інформаційних системах аудит хеш-функцій та виявлення криптографічно нестійких хеш-стрічок базується на використанні спеціалізованих програмних інструментів. Ці засоби дозволяють проводити аналіз стійкості хешів до перебору, визначати тип алгоритму, перевіряти наявність відповідності у відкритих базах витоків, а також виконувати експериментальні атаки для оцінки ризиків. У цьому підрозділі розглянуто найбільш поширені та сучасні програмні комплекси, що

застосовуються у сфері кібербезпеки, цифрової криміналістики та аудиту паролів [36].

Hashcat - високопродуктивний інструмент для криптоаналізу хешів. Hashcat є одним із найбільш швидких і функціональних рішень для перебору хешів різних форматів. Він підтримує: понад 320 типів хешів, включаючи MD5, SHA-1, SHA-2, bcrypt, scrypt, Argon2, апаратне прискорення за допомогою GPU, FPGA та ASIC, словникові, комбінаторні, регуляторні та rule-based атаки та стратегії маскування для генерації паролів зі складною структурою.

Ключовою перевагою Hashcat є масштабованість та оптимізація для паралельних обчислень**:** одна відеокарта здатна тестувати мільярди хешів за секунду, що робить інструмент корисним у рамках безпекового аудиту, але також потребує суворого дотримання етичних норм [37].

Hashcat також дозволяє ідентифікувати тип хешу за структурою та довжиною (режим --identify), що є важливим для виявлення невідомих або нестандартних форматів у системах з уразливими механізмами зберігання паролів.

John the Ripper (JtR) - універсальний інструмент із широкою підтримкою форматів. Він є одним із найбільш гнучких інструментів для аудиту хешів, який застосовується у: пентестингу; цифровій криміналістиці; перевірці корпоративних політик безпеки.

JtR підтримує численні модульні формати через "Jumbo Patch", включаючи PBKDF2, bcrypt, LM/NTLM, SHA-512-crypt тощо. Хоча його продуктивність нижча за Hashcat при використанні GPU, John the Ripper має сильний CPU-орієнтований механізм оптимізації, завдяки чому він ефективний у системах без графічних прискорювачів.

Його важливими перевагами є [38]:

- глибока система налаштувань для rule-based атак;

- можливість обробки нестандартних та самостійно створених форматів хешування;

- інтеграція з інструментами цифрової криміналістики.

Have I Been Pwned (HIBP) - сервіс перевірки хешів у базах витоків, використовується для пошуку хешованих паролів у великих відкритих базах витоків. Він застосовує модель k-анонімності, яка дозволяє надсилати лише перші 5 символів SHA-1 хешу для пошуку збігів, не розкриваючи повного значення.

Переваги HIBP:

- доступ до понад 900 млн скомпрометованих паролів;

- API для інтеграції у корпоративні системи моніторингу;

- можливість оцінити ризики використання слабких або повторно використаних паролів;

- відповідність вимогам GDPR та рекомендаціям NIST 2022 [39].

HIBP не визначає алгоритм хешування і не оцінює криптостійкість, але є незамінним інструментом для оцінювання ризику компрометації**.**

CrackStation - онлайн-інструмент для розпізнавання простих та слабких хешів, базується на великій словниковій базі та rainbow-table ресурсах, що дозволяє оперативно знаходити хеші слабких паролів.

Інструмент корисний у випадках, коли:

- потрібно швидко перевірити прості або короткі паролі;

- необхідна попередня оцінка без завантаження локального інструменту;

- досліджуються хеші, що можуть відповідати популярним фразам.

Недоліки CrackStation:

- обмеженість у роботі з криптостійкими або “повільними” алгоритмами;

- відсутність апаратної оптимізації;

- залежність від структури словника.

У рамках даного дослідження CrackStation корисний як допоміжний інструмент, але не є самостійним методом криптоаналізу [40].

Hydra / Medusa - інструменти мережевого перебору паролів, не проводять безпосереднього криптоаналізу хешів, проте часто використовуються для перевірки: стійкості паролів у мережевих сервісах (SSH, FTP, SMTP тощо), автоматизованого

перебору на основі словників та оцінювання ризиків, пов’язаних із слабкими політиками автентифікації.

Їх включення до огляду виправдане тим, що вони дозволяють виявити вторинні ризики, пов’язані з використанням слабких хеш-функцій або відсутністю сучасних механізмів зберігання паролів [41].

Порівняльна характеристика інструментів наведена у таблиці 1.1.

Таблиця 1.1

Порівняльна таблиця характеристик інструментів

Інструмент	Призначення	Переваги	Недоліки
Hashcat	перебір хешів із GPU	надвисока швидкість, велика база алгоритмів	потребує потужного обладнання
John the Ripper	CPU-криптоаналіз	універсальність, підтримка нестандартних форматів	повільніший за Hashcat
Have I Been Pwned	пошук у базах витоків	безпечна перевірка, API	не оцінює алгоритм хешування
CrackStation	словникове розпізнавання	швидкий онлайн-доступ	слабка підтримка сучасних алгоритмів
Hydra / Medusa	мережевий перебір	тестування сервісів	не аналізують хеш-функції напряму

Огляд існуючих інструментів демонструє, що жоден із них не забезпечує комплексного аналізу, що поєднує структурні, ентропійні та криптографічні характеристики хешів, а також не виконує автоматичну класифікацію типів загроз. Більшість рішень зосереджені або на переборі (Hashcat, JtR), або на перевірці витоків (HIBP), або на простих словникових атаках (CrackStation).

Це підкреслює актуальність розроблення інтелектуальної системи, яка здатна виявляти аномальні та слабкі хеш-стрічки без необхідності їхнього перебору, що і є основною метою даної кваліфікаційної роботи.

1.5. Висновки до розділу 1

У першому розділі було здійснено комплексний аналіз сучасних підходів до хешування даних, механізмів забезпечення їх криптостійкості та наявних методів виявлення вразливих хеш-значень у комп’ютеризованих системах. Розглянуті результати дозволяють сформувати узагальнену характеристику проблем та визначити ключові напрями, які потребують удосконалення.

Встановлено, що хеш-функції є базовим елементом інформаційної безпеки, оскільки вони застосовуються у зберіганні автентифікаційних даних, контролі цілісності інформації, побудові цифрових підписів та механізмах доступу. Їх стійкість визначається трьома фундаментальними властивостями: стійкістю до відновлення прообразу, second pre-image атак та колізій. Порушення будь-якої з цих властивостей призводить до суттєвого зниження рівня безпеки інформаційної системи.

Результати огляду сучасних криптографічних алгоритмів показали, що значна частина поширених хеш-функцій (MD5, SHA-1, NTLM) більше не відповідають сучасним вимогам. Незважаючи на це, вони й досі використовуються у численних корпоративних рішеннях, системах управління базами даних і спадковому програмному забезпеченні, створюючи підвищені ризики компрометації автентифікаційних даних.

Було визначено ряд типових причин виникнення слабких та аномальних хешів**,** серед яких: застосування застарілих алгоритмів, відсутність або некоректне формування «солі», конфігураційні помилки під час зберігання та обробки хешів, а також низька ентропія або невідповідність структури символьної послідовності певному алгоритму

Проведений огляд інструментів аудиту хеш-значень засвідчив, що сучасні засоби - такі як Hashcat, John the Ripper, HIBP, CrackStation - зосереджуються

переважно на переборі, словникових атаках та пошуку у базах витоків. Ці рішення ефективні для тестування стійкості або перевірки факту компрометації, однак вони не забезпечують повної автоматизованої оцінки криптоаналітичних, ентропійних та структурних властивостей хешів, а також не виявляють аномальні чи нестандартні алгоритми без виконання перебору.

Таким чином, проведений аналіз дозволяє зробити висновок, що існуючі підходи до виявлення слабких хеш-стрічок є фрагментарними та недостатньо ефективними для сучасних корпоративних систем**,** де обсяги даних стрімко зростають, а алгоритми хешування можуть бути різнорідними та неоднорідними. Це підкреслює необхідність розроблення комбінованого методу, який об’єднуватиме криптоаналітичні, статистичні та машинні підходи для всебічного аналізу хеш-значень без потреби у прямому переборі.

РОЗДІЛ 2
ТЕОРЕТИЧНА ЧАСТИНА

2.1. Математична модель структури хеш-стрічки

Криптографічні хеш-функції є ключовим механізмом забезпечення цілісності та автентичності даних у комп’ютеризованих системах. Вони перетворюють вхідну інформацію довільної довжини у символьну послідовність фіксованого розміру, яка називається хеш-значенням. У загальному випадку хеш подається у шістнадцятковому або Base64 форматі, що визначає його алфавіт та структурні характеристики [42].

Надійна хеш-функція повинна відповідати трьом фундаментальним властивостям:

- стійкості до відновлення прообразу, коли неможливо визначити початкові дані за хешем;

- стійкості до second pre-image атак, тобто неможливо підібрати інше повідомлення з тим самим хешем;

- стійкості до колізій, коли пошук двох різних повідомлень з однаковим хешем має бути обчислювально неможливим [43].

У практичних системах хеш-значення можуть втрачати свої властивості через застосування некриптографічних алгоритмів, недостатню ентропію, помилки форматування або скорочення довжини. Виявлення таких аномалій є важливим етапом оцінювання криптостійкості та формування ознак для подальшої класифікації у модулі машинного навчання.

Хеш-стрічка є результатом роботи криптографічної хеш-функції, яка перетворює вхідні дані у послідовність фіксованої довжини. Для того щоб автоматизувати процес виявлення вразливих або аномальних хешів, необхідно формалізувати їхню внутрішню структуру та виділити параметри, що піддаються кількісному аналізу.

У загальному випадку хеш-стрічку можна описати як символьну послідовність як показано у формулі (2.1):

$$h = (c_1, c_2, \ldots . . c_m), \tag{2.1}$$

де m - довжина хешу, а кожний символ c_i належить певному алфавіту, визначеному форматом подання (hex, Base64 тощо). Довжина та алфавіт є одними з найважливіших характеристик, оскільки вони дозволяють визначити потенційний тип алгоритму [45].

Для формальної моделі структури хеш-стрічки доцільно виділити три групи параметрів:

Перша група - структурні параметри. Вони описують загальні властивості хешу, такі як: довжина m, розмір алфавіту, тип форматування (hex, Base32, Base64), наявність додаткових елементів (наприклад, розділювачів або префіксів у PBKDF2 чи bcrypt).

Ці параметри дозволяють автоматично відфільтровувати неправильні, усічені або некриптографічні хеші. Структурна аномалія часто є ознакою конфігураційної помилки або використання слабкого алгоритму.

Друга група - символьні характеристики. Вони визначають розподіл і частоти символів у хеш-стрічці. До таких характеристик належать: частотні показники для кожного символу алфавіту; рівномірність розподілу символів; ймовірність появи нетипових символів.

Символьний аналіз є важливим для виявлення: некриптографічних хешів (CRC32, Jenkins тощо), хешів із низькою ентропією, нестандартних реалізацій, де розподіл символів відрізняється від очікуваного [46].

Третя група - ентропійні характеристики. Ентропія відображає ступінь випадковості символів у хеш-стрічці. Для криптографічних алгоритмів стандартом є висока ентропія, яка наближається до максимально можливої для заданого алфавіту.

Ентропійний аналіз дозволяє:

- відрізняти реальні хеші від псевдохешів або вручну створених рядків;

- виявляти пошкоджені, усічені чи частково змінені послідовності;

- виявляти наслідки помилок форматування.

Понижена ентропія може свідчити про використання слабкого алгоритму або невдалу реалізацію хешування [47].

Таким чином, математична модель хеш-стрічки базується на поєднанні трьох компонентів які показано у формулі (2.2):

$$M(h) = \big(S(h), C(h), E(h)\big), \tag{2.2}$$

де: *S(h)* - структурні характеристики;

C(h) - символьні характеристики;

E(h) - ентропійні показники.

Така модель дозволяє оцінювати хеш не лише як статичну послідовність, а як об’єкт зі складною внутрішньою структурою, що піддається формалізації та подальшій класифікації засобами машинного навчання. Узагальнена схема роботи математисної моделі хеш-стрічки наведена на рисунку 2.1.

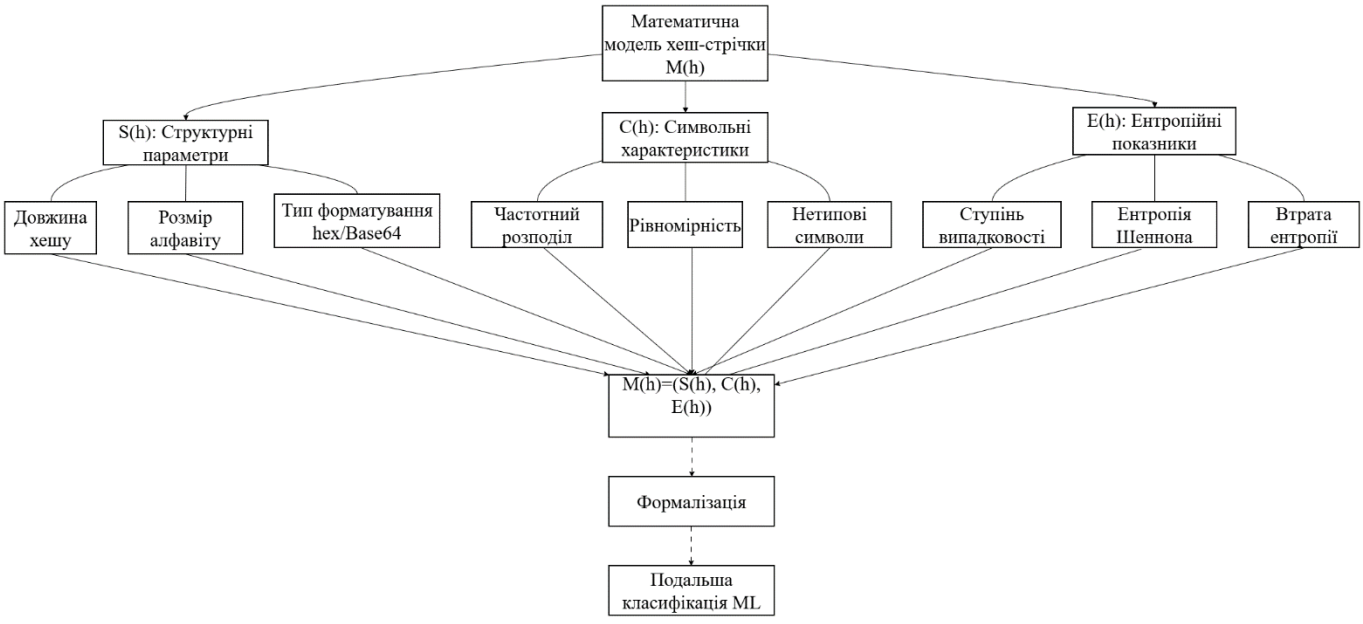

Рис 2.1. Узагальнена схема етапів математичного моделювання хеш-стрічки

2.2. Статистичні методи аналізу символьних послідовностей

Статистичний аналіз є важливим інструментом для оцінювання властивостей хеш-стрічок, оскільки дає змогу встановити ступінь випадковості, рівномірності розподілу символів і відповідність послідовності очікуваним криптографічним характеристикам. На відміну від структурного аналізу, який визначає загальні параметри хешу, статистичні методи дозволяють дослідити внутрішню поведінку символів та виявити приховані закономірності, що можуть свідчити про слабкість алгоритму або помилки реалізації [48].

Кулючові методи статистичного аналізу - це ентропія Шеннона, частотний аналіз та та оцінка символьної варіативності.

Ентропія є числовим показником ступеня випадковості символів у послідовності. Для криптографічного хешу очікується, що символи розподілені максимально рівномірно, а ентропія наближається до свого теоретичного максимуму.

Ентропія хеш-стрічки обчислюється на основі ймовірностей появи символів в алфавіті. Якщо послідовність має низьку ентропію, це може означати:

- використання некриптографічного алгоритму,
- скорочення хешу,
- повторення символів у шаблонній формі,
- неякісну генерацію випадкових даних.

Дослідження показують [49], що хеші алгоритмів MD5, SHA-1, SHA-2 та SHA-3 демонструють різну поведінку в ентропійному просторі, і ці відмінності можна використовувати як ознаки для класифікації.

Символьні характеристики хешу включають аналіз частот появи окремих символів, співвідношення між цифрами та літерами (для hex-представлення), присутності/відсутності нетипових символів та відхилення від рівномірного розподілу.

У криптографічних алгоритмах символи повинні розподілятися рівномірно, без очевидних повторюваних патернів. Виявлення будь-яких нерівномірностей може

свідчити про дефекти реалізації, викривлення даних, неавтентичні або вручну згенеровані хеші та алгоритми, що не відповідають криптографічним стандартам.

Символьний аналіз є базою для виявлення аномалій, які не можуть бути розпізнані традиційними інструментами аудиту.

Оцінка варіативності символів.

Варіативність символів показує, наскільки різноманітними є символи в межах хешу. Низька варіативність означає, що символи повторюються частіше, ніж це очікується для криптографічно стійкої функції.

Показники варіативності використовують для:

- виявлення хешів, сформованих слабкими псевдовипадковими генераторами;
- виявлення аномалій у структурі Base64-рядків;
- аналізу некриптографічних або саморобних алгоритмів.

Підвищений або занижений рівень варіативності часто корелює зі зниженою ентропією, що дозволяє комбінувати ці показники в єдину статистичну модель для подальшої класифікації [50].

Статистичні методи аналізу символьних послідовностей дозволяють створити аналітичний профіль хеш-стрічки, який враховує: ступінь випадковості; рівномірність розподілу символів; частотні закономірності; потенційні аномалії та відхилення від нормального криптографічного поведінкового патерну.

Отримані статистичні показники слугують основою для формування векторів ознак, що використовуються в алгоритмах машинного навчання при класифікації вразливих хешів. Це робить статистичний аналіз критично важливою частиною побудови системи виявлення вразливих хеш-стрічок. Узагальнена схема роботи статистичних методів аналізу символьних послідовностей наведена на рисунку 2.2.

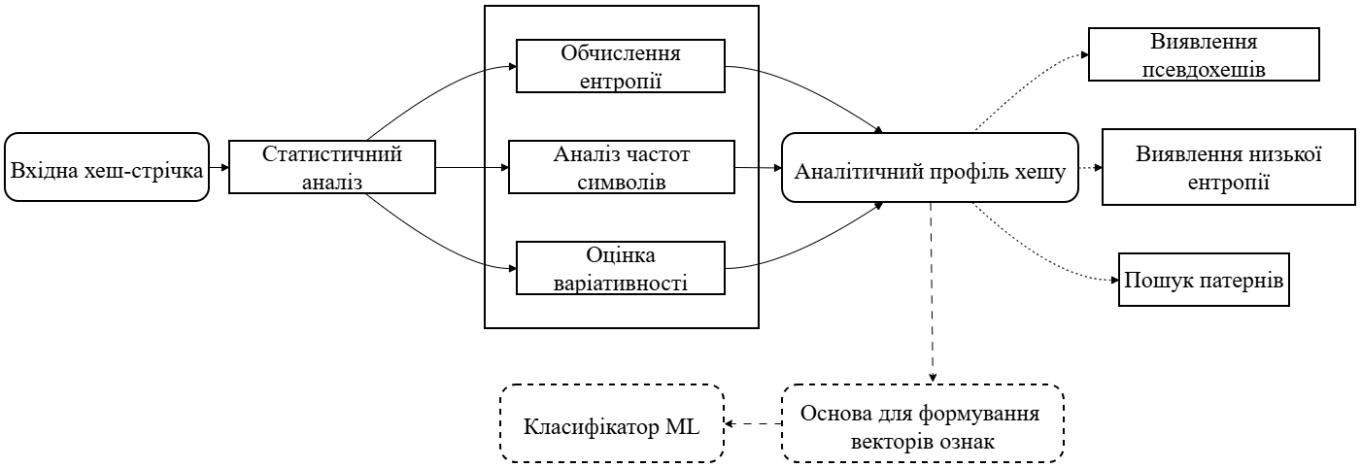

Рис. 2.2. Статистичні методи аналізу

2.3. Формування вектору ознак для класифікації хеш-значень

Для побудови ефективної системи автоматизованого виявлення вразливих хеш-стрічок необхідно сформувати вектор ознак, який однозначно та інформативно описуватиме властивості хешу. Вектор ознак (feature vector) є ключовим елементом будь-якої моделі машинного навчання, оскільки саме він визначає, які характеристики вхідних даних будуть використані для класифікації.

У випадку хеш-функцій ці ознаки повинні відображати структурні, статистичні та контекстні параметри, що дозволяють відрізняти надійні хеші від слабких, згенерованих застарілими алгоритмами або неправильно сформованими механізмами [51].

Залежно від типу характеристики, весь набір ознак можна поділити на такі основні групи - структурні ознаки, символьні ознаки, ентропійні ознаки, алгоритмічні та контексні ознаки.

Структурні параметри є базовими характеристиками рядка хешу і дозволяють визначити його відповідність типу алгоритму. До цієї групи належать:

- довжина хешу (m) (одна з найважливіших ознак, оскільки більшість алгоритмів мають строго фіксовану довжину);

- тип алфавіту (hex, Base32, Base64);

- наявність нестандартних символів (помилки або модифікації формату);

- наявність структурних маркерів;

- визначення, чи може бути «сіль» та її орієнтовна довжина, якщо рядок містить маркери.

Структурні ознаки дозволяють одразу класифікувати значну частину хешів без складних обчислень і є ключовими у попередньому фільтруванні [52].

Символьні ознаки визначають розподіл та властивості символів у хеш-стрічці. До них належать:

- частотний розподіл символів (digits/letters);

- частка повторюваних символів;

- співвідношення великих і малих літер (для Base64);

- кількість унікальних символів;

- відхилення від рівномірності розподілу.

Такі ознаки важливі для виявлення:

- некриптографічних функцій, де розподіл символів може бути нерівномірним;

- пошкоджених або вручну модифікованих хешів;

- аномалій у реалізації алгоритму.

Ентропійні ознаки.

Ентропія є ключовим показником криптографічної якості хешу. Вектор ознак може включати такі параметри:

- ентропія Шеннона для всієї послідовності;

- локальна ентропія окремих сегментів;

- ентропійні відхилення від теоретичного максимуму для даного алфавіту;

- коефіцієнт варіативності символів.

Низька ентропія може сигналізувати про: слабку генерацію випадкових значень, використання простих хешів та некоректні кастомні реалізації.

Ентропійні ознаки є важливими в алгоритмах машинного навчання і добре підходять для класифікації нестандартних або підозрілих хешів [53].

Контекстні ознаки формуються на основі аналізу формату та характерних шаблонів, специфічних для певних алгоритмів. Вони включають:

- припущений тип алгоритму (на основі довжини та маркерів);

- ймовірність відповідності певному алгоритму (обчислювана евристично);

- наявність структур, характерних для PBKDF2 (ітерації), bcrypt (cost factor), Argon2 (memory/time parameters**)**;

- наявність префіксів або суфіксів, що містять службову інформацію**.**

Такі ознаки дозволяють моделі ML: розрізняти криптостійкі алгоритми від застарілих, визначати, чи була «сіль» застосована правильно та розпізнавати складні та комбіновані формати.

На основі описаних груп формується вектор як показано у формулі (2.3):

$$F(h) = (f_1, f_2, \dots, f_k), \tag{2.3}$$

де f_k- це окрема числова або категоріальна характеристика хеш-стрічки.

Загалом для ефективної класифікації зазвичай формується від 20 до 60 ознак, залежно від глибини моделі. Такі ознаки дозволяють моделі машинного навчання розпізнавати навіть неочевидні закономірності, які неможливо виявити традиційними методами аудиту. Правильно сформований вектор ознак забезпечує: точність класифікації; здатність моделі відрізняти нові, невідомі алгоритми; підвищення стійкості системи до хибнопозитивних результатів; можливість розширення системи без зміни основної логіки. Це створює математичне підґрунтя для побудови комбінованого методу виявлення вразливих хешів. Узагальнена схема роботи формування вектора ознак для класифікації хеш значень наведена на рисунку 2.3.

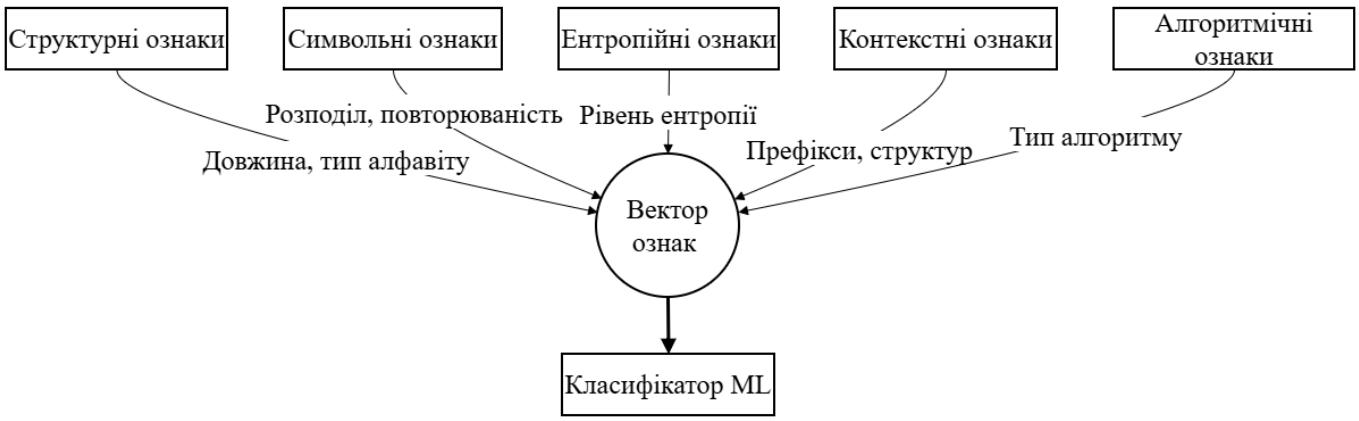

Рис 2.3. Схема формування вектора ознак для класифікації хеш-значень

2.4. Методи машинного навчання для класифікації вразливих хеш-стрічок.

Застосування машинного навчання у задачі класифікації хеш-стрічок відкриває можливість автоматизовано визначати слабкі, аномальні та криптографічно нестійкі хеші без необхідності виконання криптоаналізу або перебору. На основі сформованого вектора ознак моделі ML можуть виявляти приховані закономірності, що не піддаються простому правилам-аналітичному аналізу [54].

У цьому підпункті розглядаються алгоритми, які показали найкращу ефективність у задачах класифікації символьних послідовностей з високою варіативністю: Random Forest**,** XGBoost та додатково - логістична регресія як базова модель.

Random Forest є ансамблевим методом, що об’єднує велику кількість дерев рішень, кожне з яких навчається на випадковій підмножині ознак і даних. Таке поєднання знижує ризик перенавчання та забезпечує високу стабільність результатів.

Переваги Random Forest у контексті аналізу хешів:

- добре працює з числовими та категоріальними ознаками;
- стійкий до шумів у даних;
- дозволяє оцінити важливість ознак (feature importance);
- ефективний навіть при невеликому обсязі навчальної вибірки.

Для задачі визначення типу алгоритму (MD5, SHA-1, SHA-256, bcrypt) Random Forest демонструє точність понад 95% при правильно сформованому наборі характеристик [55].

XGBoost (Extreme Gradient Boosting) є одним із найпотужніших алгоритмів градієнтного бустингу та широко застосовується для класифікації у кібербезпеці. Його основна ідея полягає в послідовному навчанні дерев рішень, де кожне наступне дерево виправляє помилки попередніх.

Переваги XGBoost:

- висока точність класифікації, особливо для складних ознак;
- підтримка регуляризації, що захищає модель від перенавчання;
- здатність виявляти слабкі, нестандартні та рідкісні формати хешів;

- швидкість тренування завдяки оптимізованій реалізації.

У багатьох дослідженнях XGBoost перевершує Random Forest у задачах розпізнавання криптографічних аномалій завдяки своїй здатності моделювати складні нелінійні залежності [56].

Логістична регресія використовується як проста, але ефективна базова модель для задач двокласової та багатокласової класифікації. У контексті хешів вона є корисною:

- як відправна точка для оцінювання якості ознак;

- для інтерпретованих моделей, коли важливо зрозуміти, які фактори впливають на класифікацію;

- у випадках, коли дані мають лінійно відокремлювану структуру.

Хоча логістична регресія зазвичай поступається ансамблевим алгоритмам у точності, вона дає можливість швидко протестувати модель та оцінити актуальність сформованих ознак.

Для оцінювання ефективності моделей машинного навчання застосовують кілька ключових метрик:

- Accuracy (загальна частка правильних передбачень);

- Precision та Recall (важливі у задачах із можливими хибнопозитивними результатами);

- F1-score (гармонійне середнє між precision і recall);

- ROC AUC (площа під ROC-кривою, показує здатність моделі розділяти класи).

У контексті виявлення вразливих хешів високий recall є критично важливим, оскільки пропущені слабкі або аномальні хеші можуть становити реальну загрозу безпеці системи.

Методи машинного навчання забезпечують:

- адаптивне виявлення слабких та аномальних хешів;

- здатність обробляти великі обсяги даних без ручного аналізу;

- високу точність класифікації при правильному формуванні ознак;

- гнучкість при додаванні нових алгоритмів хешування.

Ансамблеві алгоритми (Random Forest, XGBoost) показують найкращі результати й будуть використані як основа для розроблення класифікаційної моделі у цьому проєкті. Узагальнена схема роботи методів машиного навчання для класифікації вразливих хеш- стрічок наведена на рисунку 2.4.

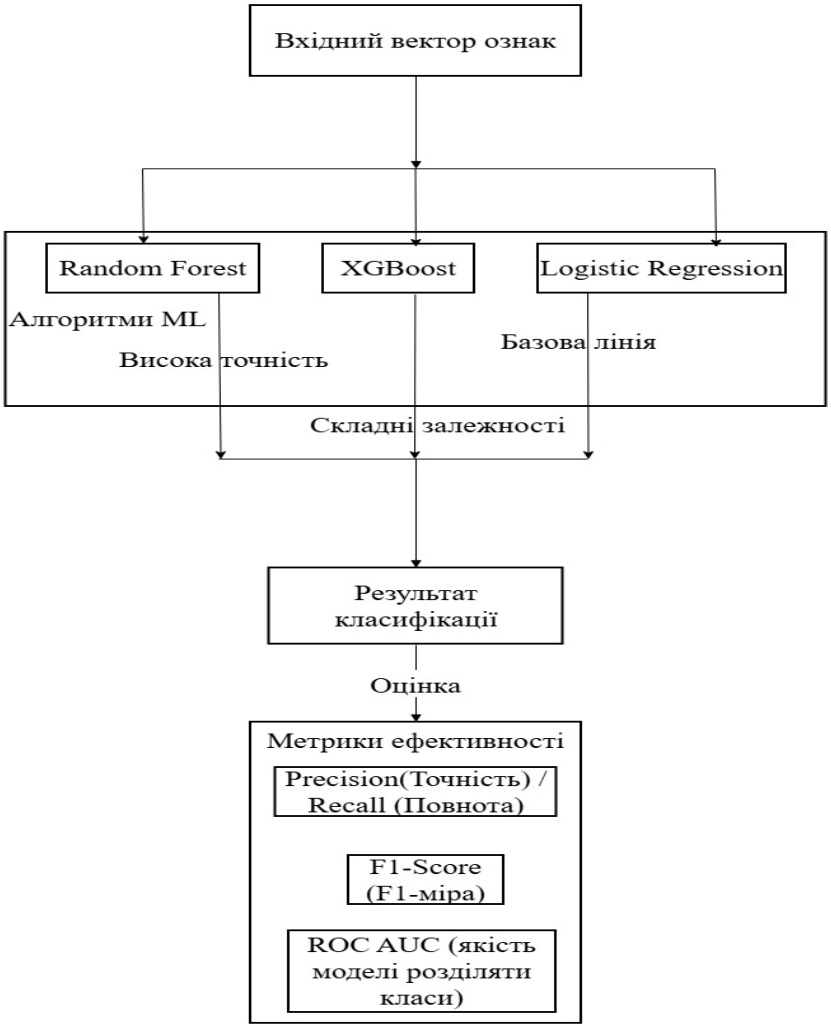

Рис 2.4. Схема роботи методів машиного навчання для класифікації вразливих хеш-стрічок

2.5. Обґрунтування вибору методу виявлення вразливих хеш-стрічок

Аналіз існуючих методів виявлення вразливих хеш-стрічок показує, що жоден окремий підхід - структурний, статистичний або машинний - не забезпечує достатнього рівня достовірності при класифікації слабких та аномальних хеш-

стрічок. Тому для вирішення поставленої задачі доцільним є застосування комбінованого методу, який об’єднує переваги різних аналітичних підходів та мінімізує їхні недоліки.

Класичні методи оцінювання стійкості хешів ґрунтуються на: перевірці довжини та структури послідовності, визначенні алгоритму за форматом, пошуку колізій та тестуванні на перебір.

Однак ці методи мають низку недоліків:

- не виявляють модифіковані, авторські або нестандартні алгоритми;
- не розпізнають пошкоджені або частково усічені хеші;
- не здатні визначити низьку ентропію або аномальний розподіл символів;
- покладаються на ручний аналіз або перебір, що не масштабуються для великих баз даних.

Сучасні дослідження підтверджують обмеженість традиційного криптоаналізу в умовах швидко зростаючих інформаційних систем [57].

Статистичні показники (ентропія, частотний розподіл, варіативність символів) дозволяють виявляти аномалії, однак вони не можуть:

- визначити конкретний тип алгоритму;
- розрізнити, чи низька ентропія спричинена слабким алгоритмом чи спотворенням даних;
- коректно обробляти комбіновані формати;
- працювати з несимвольними параметрами (ітерації, salting-поля тощо).

Отже, статистичний аналіз є важливим, але недостатнім компонентом системи [58].

Моделі ML демонструють високу ефективність у задачах класифікації лише тоді, коли сформовано інформативний та різноманітний вектор ознак. Якщо ж ознаки включають лише структурні або статистичні параметри, модель: не може коректно ідентифікувати складні криптографічні формати, може навчитися хибним закономірностям, втрачає точність у випадку нових, ще не зустрічених алгоритмів та стає нестійкою до аномалій у реальних базах даних.

Тому машинне навчання має використовуватися разом зі структурними та криптографічними характеристиками, а не замість них [59].

Запропонований комбінований метод поєднує три підходи: криптоаналітичний аналіз, статистичний та ентропійний аналіз, та модель машиного навчання.

Криптоаналітичний аналіз. Визначає структурні характеристики: довжину, формат, ознаки алгоритму, наявність «солі», префіксів та службових параметрів;

Статистичний та ентропійний аналіз. Виявляє нестандартні, слабкі або пошкоджені послідовності, які не відповідають очікуваному криптографічному розподілу;

Моделі машинного навчання. Класифікують хеш-рядки, інтегруючи результати всіх попередніх аналізів, та виявляють приховані закономірності у великій множині ознак.

Комбінований метод: підвищує точність класифікації до рівня, недосяжного окремими методами, забезпечує стабільність роботи при наявності шумів і аномалій у даних, здатний розпізнавати нові та рідкісні формати хешів**,** дозволяє виключити необхідність перебору або криптоатаки та пропонує масштабованість для великих інформаційних систем.

Наукові публікації підтверджують [57–59], що інтеграція криптографічних і статистичних ознак із ML-моделями дає найкращі результати у задачах класифікації хешів та виявлення аномалій.

Запропонований метод формує основу архітектури системи, оскільки: дозволяє автоматично обробляти різні типи хешів, забезпечує високу точність навіть на реальних базах даних, де можливі помилки, легко адаптується для нових алгоритмів і форматів та забезпечує модульність та зрозумілу інтерпретацію результатів.

Таким чином, комбінований підхід є оптимальним вирішенням задачі виявлення вразливих хеш-стрічок і повністю відповідає цілям уваліфікаційної роботи магістра. Узагальнена робота комбінованого методу виявлення вразливих хешів наведена на рисунку 2.5.

Рис 2.5. Схема роботи комбінованого методу виявлення вразливих хешів

2.6. Висновки до розділу 2

У другому розділі було сформовано теоретичне та методичне підґрунтя для розроблення системи автоматизованого виявлення вразливих хеш-стрічок. Розглянуті підходи дали змогу визначити ключові математичні властивості хеш-функцій, описати структурні та статистичні параметри хеш-значень і сформувати вимоги до моделі аналізу.

Показано, що хеш-стрічка є не лише символьною послідовністю фіксованої довжини, але й носієм важливої внутрішньої інформації, яка відображає принцип роботи алгоритму хешування. Завдяки формалізації структури хешу та виділенню його основних характеристик стало можливим визначити групи ознак, придатних для автоматизованої класифікації.

Було проаналізовано статистичні методи, зокрема ентропійний аналіз, частотні та символьні показники, які дозволяють виявляти аномалії, властиві некриптографічним або спотвореним хешам. Ентропія та варіативність символів показали особливу цінність як універсальні маркери якості хешування.

На основі структурних, символьних і ентропійних характеристик сформовано вектор ознак, придатний для використання моделей машинного навчання. Показано, що об’єднання різнорідних ознак підвищує інформативність моделі та забезпечує коректну класифікацію хешів різного походження.

Проведено порівняння алгоритмів машинного навчання, придатних для аналізу хеш-стрічок, і встановлено, що ансамблеві методи (Random Forest, XGBoost) демонструють найкращу точність та стабільність. Базові моделі, такі як логістична регресія, можуть використовуватися як орієнтовний інструмент для перевірки якості ознак.

Обґрунтовано доцільність застосування комбінованого підходу, який поєднує криптоаналітичний, статистичний та машинний аналіз. Саме така інтеграція дозволяє ефективно виявляти слабкі, застарілі та аномальні хеш-стрічки.

РОЗДІЛ 3
ПРАКТИЧНА ЧАСТИНА

3.1. Постановка задачі розроблення комп’ютеризованої системи

На основі проведеного аналізу та сформульованих теоретичних засад виникає необхідність створення системи виявлення вразливих хеш-стрічок, здатної автоматизовано виявляти вразливі хеш-стрічки у комп’ютеризованих системах. Зростання обсягів даних, різноманіття алгоритмів хешування та можливість появи аномальних або некоректно сформованих хешів вимагають інтегрованого рішення, яке поєднує криптоаналітичні, статистичні й машинні методи оцінювання.

Запропонована система виявлення вразливих хеш-стрічок - забезпечить оперативну, точну та масштабовану ідентифікацію хешів, що мають ознаки криптографічної слабкості**,** а також визначить потенційні загрози для політик автентифікації у корпоративних середовищах. На відміну від традиційних інструментів, орієнтованих переважно на перебір чи перевірку у базах витоків, система має виконувати аналітичне оцінювання хешу без відновлення вихідного повідомлення**.**

Система повинна мати універсальний механізм прийому даних з різних джерел: бази даних, файли конфігурацій та логів, API корпоративних систем.

Отримані дані мають бути нормалізовані, перевірені на коректність і підготовлені до аналізу.

Система повинна: аналізувати довжину, формат та алфавіт хешу, визначати наявність структурних маркерів алгоритмів (bcrypt, PBKDF2, Argon2) та виявляти некоректні або усічені хеші.

Це створює базове підґрунтя для подальшої класифікації.

На етапі виконання статистичного та ентропійного аналізу система має: оцінювати рівномірність розподілу символів, вимірювати ентропію послідовності та

виявляти аномалії, які можуть вказувати на слабкий або некриптографічний алгоритм.

Статистичний аналіз дозволяє перевірити хеш незалежно від його оголошеного алгоритму.

Система повинна використовувати модель ML, тренуваня на структурних, символьних та ентропійних ознаках, для визначення: чи є хеш криптостійким, чи належить він до застарілого алгоритму та чи має ознаки аномальності або помилок формування.

Модель має забезпечувати точність, масштабованість та можливість подальшого навчання.

Система повинна надавати: висновок про криптографічну якість хешу, ймовірність належності до певного алгоритму та рекомендації щодо усунення вразливості.

Додатково необхідно передбачити: REST API для взаємодії з зовнішніми системами, можливість формування звітів і логування та аудит операцій для корпоративних потреб.

Таким чином, розроблювана система повинна об’єднувати:

- структурний аналіз,
- ентропійні та статистичні методи,
- алгоритми машинного навчання,
- механізми інтеграції та автоматизації.

Це дозволить виявляти вразливі, застарілі чи аномальні хеш-стрічки без потреби у переборі, що робить систему ефективним інструментом для сучасних корпоративних і веб-інфраструктур.

3.2. Архітектура системи

На основі сформульованих вимог було розроблено архітектуру системи, яка забезпечує комплексний аналіз хеш-стрічок і визначення їх криптографічної стійкості. Архітектурне рішення побудовано відповідно до принципів модульності,

розширюваності та незалежності компонентів, що дає змогу адаптувати систему до різних технічних середовищ і джерел даних.

Система складається з п’яти основних модулів:

- модуль приймання та нормалізації даних;
- криптоаналітичний модуль;
- модуль статистичного та ентропійного аналізу;
- модуль машинного навчання;
- інтерфейс взаємодії та підсистема звітності.

На рисунку 3.1 показана структурна схема архітектури системи.

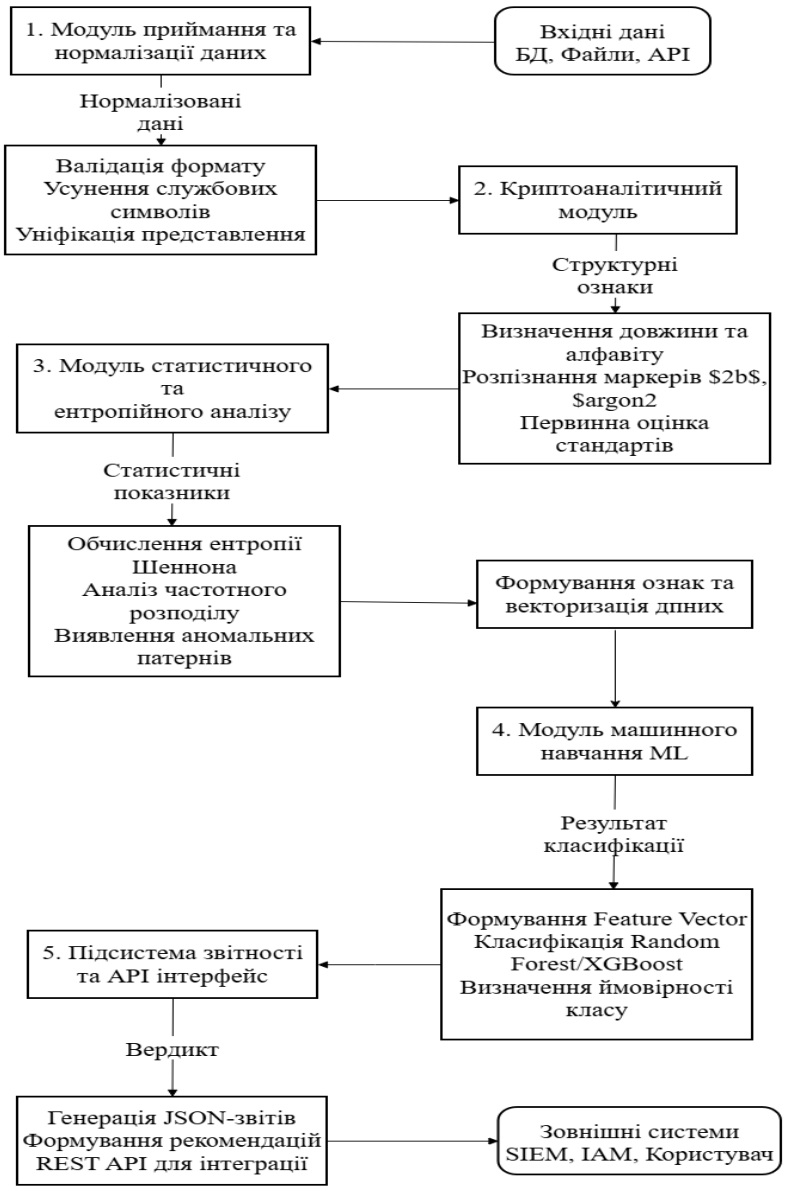

Рис 3.1. Структурна схема архітектури системи

Кожен модуль виконує окрему функцію, але працює у межах узгодженого конвеєра обробки.

Модуль приймання та нормалізації даних відповідає за: отримання хешів з різних джерел (БД, файли, журнали подій, API), валідацію вхідної інформації (перевірка формату, усунення службових символів), перетворення даних у внутрішнє уніфіковане представлення та фільтрацію некоректних або порожніх записів.

Завдяки цьому системі не потрібно адаптуватися під кожний тип бази даних окремо - модуль забезпечує незалежність і чистоту даних на вході.

Криптоаналітичний модуль призначений для: визначення довжини хешу, аналіз алфавіту (hex, Base32, Base64), виявлення структурних маркерів алгоритмів, автоматичне розпізнання формату PBKDF2, bcrypt, Argon2 та первинної оцінки криптостійкості на основі відповідності сучасним стандартам.

Цей модуль функціонує як первинний фільтр: він відсіює рядки, що не є хешами, та визначає базові характеристики, необхідні для подальшого аналізу.

Модуль статистичного та ентропійного аналізу виконує поглиблений аналіз символьної послідовності:

- обчислення ентропії Шеннона;
- аналіз частотного розподілу символів;
- виявлення нетипових або рідкісних патернів;
- визначення рівня варіативності символів;
- порівняння отриманих показників із еталонними для конкретних алгоритмів.

Якщо хеш має занижену ентропію, аномальні повтори символів або неправильну структуру, модуль позначає його як потенційно вразливий.

Модуль машинного навчання - це центральний компонент архітектури. Він отримує набір ознак (feature vector), сформований із: структурних параметрів, статистичних показників та криптоаналітичних характеристик.

У модулі реалізовано: механізм передобробки ознак, навчальну модель (Random Forest або XGBoost), інтерфейс для донавчання моделі, механізм обчислення ймовірності належності хешу до класів а саме (слабкий, аномальний, криптостійкий, застарілий алгоритм).

Модуль ML дозволяє виявляти аномалії, які неможливо визначити за допомогою традиційного криптоаналізу.

Підсистема звітності та API інтерфейс має такі функції:

- генерація структурованих звітів про стан хеш-стрічок;
- формування рекомендацій щодо усунення вразливості;
- запис логів та результатів аналізу;
- REST API для інтеграції з корпоративними системами моніторингу, IAM-платформами або SIEM-рішеннями;
- підтримка формату JSON для обміну даними.

Ця підсистема робить розроблену систему придатною для реального промислового використання.

Взаємодія модулів:

- дані надходять до модуля нормалізації;
- криптоаналітичний модуль визначає ключові параметри хешу;
- статистичний модуль оцінює ентропію та символьні характеристики;
- модуль ML класифікує хеш та визначає рівень ризику;
- підсистема звітності передає результати користувачу або зовнішній системі.

Таким чином, архітектура розроблюваної системи побудована у вигляді послідовного конвеєра, який забезпечує повний цикл аналізу хеш-значень.

3.3. Реалізація криптоаналітичного модуля

Криптоаналітичний модуль є початковим та одним із ключових компонентів системи, оскільки саме він виконує базову ідентифікацію та структурний аналіз хеш-стрічок до застосування статистичних та інтелектуальних методів. Основне призначення цього модуля полягає у визначенні можливого алгоритму хешування, оцінюванні відповідності формату сучасним криптографічним стандартам, виявленні структурних аномалій та формуванні початкового профілю хешу для подальшої обробки ентропійним та ML-модулями. Схема роботи криптоаналітичного модуля наведена на рисунку 3.2.

Рис 3.2. Схема роботи криптоаналітичного модуля

Першим етапом роботи модуля є аналіз довжини та синтаксичної структури вхідної хеш-стрічки. Для більшості поширених алгоритмів хешування характерні фіксовані або очікувані розміри вихідних значень. Наприклад, MD5, SHA-1, SHA-256, SHA-512, NTLM мають чітко визначену довжину шістнадцяткового подання, тоді як адаптивні алгоритми на кшталт bcrypt, PBKDF2, Argon2 містять додаткові службові сегменти, параметри складності та маркери формату.

На цьому етапі модуль: визначає фактичну довжину хешу як символьного рядка, аналізує наявність префіксів ($2b$, $argon2id$, pbkdf2_ тощо), перевіряє наявність розділювачів ($, :, інші спеціальні символи), виявляє потенційні ознаки

усічення (несподівано короткі рядки, «обірвані» структури) та визначає, чи може хеш належати до відомої групи алгоритмів.

Якщо за сукупністю ознак хеш не відповідає жодному із відомих форматів, він позначається як структурно аномальний і передається на подальший аналіз із підвищеним рівнем ризику.

Другим етапом є визначення типу алфавіту, який використовується для подання хешу. Різні алгоритми застосовують шістнадцяткове подання, Base32, Base64, а також спеціальні алфавіти (наприклад, bcrypt-алфавіт).

Криптоаналітичний модуль:

- аналізує, чи складається хеш лише з допустимих hex-символів (0–9, A–F);

- перевіряє, чи відповідає рядок алфавіту Base64 (літери, цифри, +, /, =);

- ідентифікує спеціальні символи, характерні для bcrypt та Argon2;

- визначає, чи спостерігається змішаний або нетиповий алфавіт, що може свідчити про некоректну реалізацію.

Додатково модуль шукає характерні структурні патерни (наявність декількох частин, розділених : або $), що дозволяє точніше ідентифікувати алгоритм навіть при частковому пошкодженні вихідних даних.

Після попередньої ідентифікації типу та структури хешу модуль виконує оцінювання криптографічної придатності алгоритму. На основі сучасних рекомендацій міжнародних стандартів та актуальних оглядів індустрії. Алгоритми умовно поділяються на такі групи:

- сучасні та рекомендовані: SHA-256, SHA-3, bcrypt, Argon2, PBKDF2;

- умовно допустимі (у вузьких сценаріях): SHA-1 для некритичних задач;

- застарілі та небезпечні: MD5, SHA-1 для автентифікації, NTLM, LM-хеші.

На цьому етапі модуль:

- фіксує попередню оцінку рівня ризику для кожного хешу (низький, середній, високий);

- відзначає відсутність або наявність «солі»;

- аналізує параметри адаптивних алгоритмів (кількість ітерацій у PBKDF2, cost factor у bcrypt, параметри пам’яті та часу в Argon2, якщо вони доступні із рядка);

-виявляє можливі ознаки ручної модифікації (нестандартні префікси, відсутні сегменти, нетипові символи).

Формування профілю хешу

Результатом роботи криптоаналітичного модуля є профіль хешу (Hash Profile) - структурований набір полів, що описує кожну хеш-стрічку з точки зору її:

- ймовірного алгоритму хешування;
- типу алфавіту (hex, Base64, нестандартний);
- синтаксичної структури (кількість сегментів, префікси, роздільники);
- попередньої оцінки рівня ризику;
- виявлених аномалій (усічення, нестандартні символи, підозрілий формат);
- рекомендацій щодо подальшої обробки.

Цей профіль передається як вхідні дані до ентропійного модуля та ML-класифікатора. Фрагмент програмної реалізації формування профілю хешу в системі наведено у додатку Б, де показано практичноу реалізацію криптоаналітичного модуля на мові Python.

Місце криптоаналітичного модуля в архітектурі системи.

У загальній архітектурі системи криптоаналітичний модуль виконує роль первинного фільтра та нормалізатора вхідних даних. Від точності його роботи залежить коректність статистичного аналізу, якість сформованого вектора ознак та, відповідно, ефективність навченої ML-моделі.

Основні функції модуля в контексті архітектури:

- прийом вхідних хеш-стрічок від підсистеми доступу до даних;
- нормалізація та попередня перевірка вхідних значень;
- структурний, синтаксичний та криптографічний аналіз;
- формування профілю хешу та передача його до наступних підсистем;
- журналювання виявлених аномалій для подальшого аудиту.

На схемі роботи модуля відображено основні кроки обробки, починаючи від отримання хешу та визначення його структури до формування повного профілю, який надходить на вхід ентропійного аналізу та ML-класифікатора.

3.4. Реалізація модуля ентропійного аналізу

Модуль ентропійного аналізу є другим основним компонентом системи, який виконує поглиблену статистичну оцінку хеш-значення після первинної обробки криптоаналітичним модулем. Його завдання полягає у визначенні ступеня випадковості символів, виявленні структурних відхилень, оцінюванні характеру символьних розподілів та формуванні числових ознак, що використовуються в моделі машинного навчання. На цьому етапі хеш аналізується з точки зору статистики інформації, що дозволяє виявляти слабкі або аномальні значення, які не завжди можна розпізнати суто за формальними правилами формату. Алгоритм роботи модуля ентропійного аналізу наведена на рисунку 3.3.

Ентропійний модуль визначає, наскільки символи хеш-рядка є випадковими або структурно обмеженими. Зазвичай коректно сформовані криптографічні хеші характеризуються високою рівномірністю розподілу символів. Якщо ж у значенні наявні повтори, закономірності або ненормальна концентрація певних символів, це може свідчити про:

- використання нестійкого або старого алгоритму;
- неправильну чи спрощену реалізацію хешування;
- некоректну конкатенацію або форматування хеш-рядків;
- спроби ручної модифікації або кодування нестандартними методами [70].

Таким чином, ентропійні характеристики суттєво доповнюють структурний аналіз і підвищують точність класифікації.

У модулі реалізовано обчислення кількох ключових статистичних ознак, які показали ефективність у виявленні аномальних хешів:

- ентропія Шеннона (оцінка рівня випадковості символів у хеш-стрічці);
- частотний розподіл символів (виявлення надлишкових повторів або перекошеного алфавітного профілю);
- співвідношення унікальних до повторюваних символів (корисний критерій для виявлення хешів, згенерованих слабкими або неправильними алгоритмами);

- визначення домінуючих підпослідовностей (реєстрація підозрілих повторюваних фрагментів, характерних для "ручних" або нестандартних методів хешування);

- символьні аномалії (виявлення змішаних алфавітів, порожніх сегментів, а також фрагментів, що не характерні для криптографічних форматів).

Рис 3.3. Алгоритм роботи модуля ентропійного аналізу

Дані параметри формують основу для подальшої класифікації, збагачуючи вектор ознак системи.

Послідовність обробки в модулі ентропійного аналізу передбачає такі кроки:

- отримання профілю хешу від криптоаналітичного модуля;

- обчислення ентропійних та статистичних характеристик на основі символьного складу.

Формування структурованих ознак (entropy profile) які включають: числові метрики ентропії, кількість унікальних і повторюваних символів, коефіцієнти символьної рівномірності та показники розподілу алфавіту.

Структура роботи модуля відповідає вимогам модульності та забезпечує незалежність від конкретного алгоритму хешування, що є важливим для застосування у великих гетерогенних системах. Реалізація ентропійного аналізу у Python наведено у додатку Б.

У наведеному фрагменті коду продемонстровано базову реалізацію модуля ентропійного аналізу. Він обчислює ентропію Шеннона, частоти символів і формує набір числових ознак, які потім використовуються ML-класифікатором.

Модуль ентропійного аналізу відіграє важливу роль у виявленні підозрілих або нетипових хешів, які можуть не викликати підозри за формальними структурними ознаками. Його результати:

-збагачують вектор ознак, який передається в ML-модель;

- дозволяють виявляти хеші зі зниженою випадковістю символів (ознаки слабких алгоритмів);

- допомагають оцінити коректність реалізації хешування;

- підвищують точність класифікації, особливо у випадках змішаних або нестандартних форматів.

Таким чином, модуль ентропійного аналізу забезпечує додатковий рівень контролю та детальний статистичний огляд хеш-даних перед їхньою інтелектуальною обробкою.

3.5. Формування ознак та векторизація даних

Ефективність роботи моделі машинного навчання у системі значною мірою залежить від якості сформованого вектору ознак, який містить структурну, ентропійну та контекстну інформацію про кожну хеш-стрічку. Саме на цьому етапі результа роботи криптоаналітичного та ентропійного модулів інтегруються у єдине числове подання, придатне для класифікації. Формування набору ознак дозволяє перетворити текстове хеш-значення на вектор, що репрезентує статистичні, семантичні та структурні характеристики хешу у форматі, який може бути опрацьований ML-класифікатором.

Основною задачею векторизації є:

- перетворення символьного хеш-рядка на набір кількісних параметрів;
- інтеграція структурних ознак, отриманих криптоаналітичним модулем;
- включення статистичних та ентропійних характеристик;
- уніфікація різних форматів хешів у спільний тип даних;
- підготовка входу для ML-моделі відповідно до вибраної архітектури (Random Forest, XGBoost тощо).

Сформований вектор ознак повинен бути інформативним, стійким до шумів, але водночас достатньо компактним для уникнення перенавчання.

На основі експериментальних досліджень та оглядів сучасних методів класифікації криптографічних об’єктів було визначено оптимальний набір ознак, які демонструють високу інформативність у задачі виявлення вразливих хешів.

Структурні ознаки (Crypto Features):

- hash_length (довжина хеш-стрічки);
- alphabet_type (тип алфавіту (hex, Base64, bcrypt, змішаний);
- segment_count (кількість сегментів, розділених $, :, =);
- has_prefix (наявність службових префіксів);
- salt_present (ознака наявності «солі»);
- algorithm_guess (ймовірний алгоритм).

Ентропійні ознаки (Entropy Features):

- shannon_entropy (ентропія Шеннона);

- unique_char_count (кількість унікальних символів);

- repeat_ratio (частка повторюваних символів);

- symbol_distribution (частотний вектор для символів).

Аномальні ознаки (Anomaly Indicators):

- is_truncated (підозра на усічення);

- irregular_pattern_detected (ознака наявності повторюваних фрагментів);

- mixed_alphabet_flag (використання нетипового алфавіту);

- rare_symbols_flag (наявність символів, які не властиві жодному криптографічному алгоритму).

Поєднання цих ознак дозволяє значно підвищити точність класифікації, особливо у випадках, коли хеш не має чіткої структурної ідентифікації або містить аномалії, непомітні для традиційних сигнатурних методів. Формування ознак наведено на рисунку 3.4.

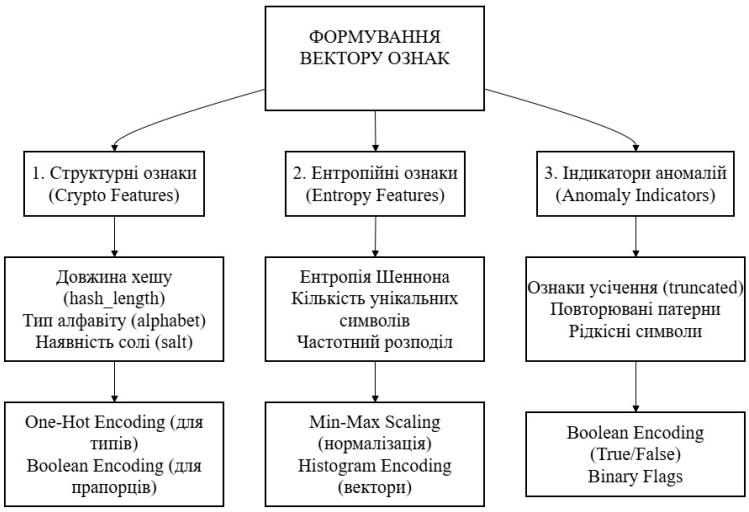

Рис 3.4. Формування ознак

Залежно від типу ознаки в системі застосовуються різні методи векторизації:

- one-Hot Encoding (для категоріальних параметрів: тип алгоритму, тип алфавіту).

- min-Max Scaling або Standard Scaling (для числових ознак:довжина, ентропія).

- normalized Histogram Encoding (для частотного розподілу символів).

- boolean Encoding (для бінарних ознак: наявність «солі», структурні аномалії).

Усі ознаки об’єднуються в єдиний масив (feature vector), який має фіксовану структуру незалежно від типу вхідного хешу.

Процес формування ознак складається з таких кроків:

- отримання HashProfile та EntropyProfile з попередніх модулів;

- виділення структурних, ентропійних та бінарних ознак;

- нормалізація та кодування значень;

- формування фінального вектору ознак у форматі NumPy або Pandas Series;

- передавання сформованого вектору до ML-модуля для класифікації.

Алгоритм роботи модуля векторизації даних наведена на рисунку 3.5.

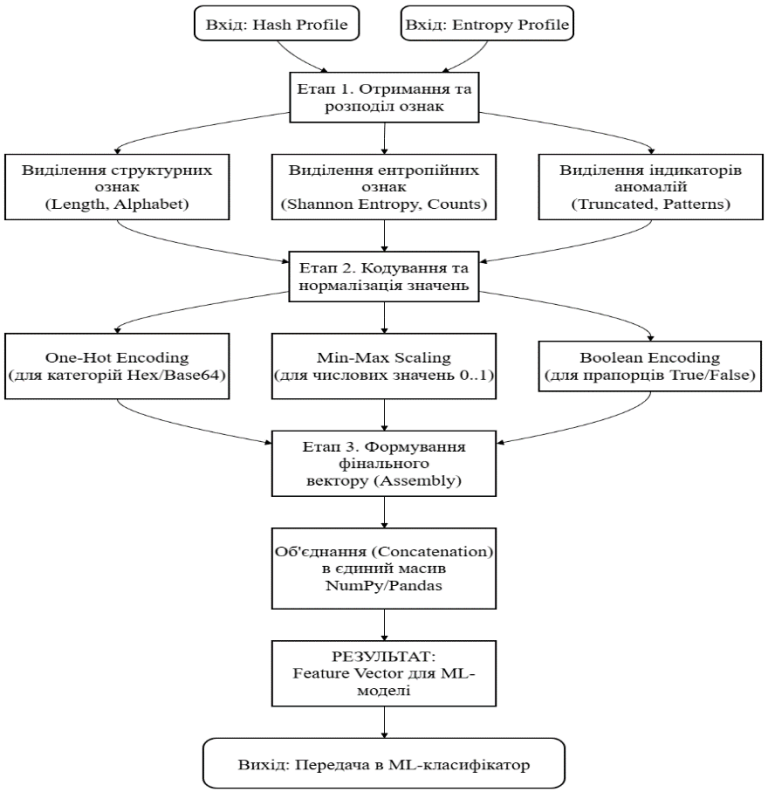

Рис 3.5 Алгоритм роботи векторизації даних

Цей підхід забезпечує сумісність між різними компонентами системи та дозволяє масштабувати модель до великих обсягів даних. Фрагмент програмної реалізації векторизації наведено у додатку Б.

Модуль векторизації виконує критично важливу функцію - створення узагальненого числового представлення хешу, яке здатне відобразити:

- криптографічні властивості;
- статистичні характеристики;
- структурні та символьні відхилення.

Якість і повнота цього представлення впливають на результат роботи ML-класифікатора, оскільки забезпечують його релевантними ознаками для навчання та прогнозування.

3.6. Розроблення ML-модуля

Після формування повного вектору ознак система переходить до етапу використання моделі машинного навчання, яка виконує класифікацію хеш-значень за рівнем криптографічної стійкості та можливими структурними аномаліями. Метою розроблення ML-модуля є створення надійного інтелектуального механізму, здатного автоматично розрізняти:

- стійкі сучасні хеші;
- хеші, сформовані застарілими або небезпечними алгоритмами;
- аномальні та некоректно побудовані хеш-стрічки;
- результати псевдохешування чи нестандартних реалізацій.

ML-модуль є ключовим компонентом системи, який реалізує інтелектуальну частину системи та забезпечує автоматичну інтерпретацію ознак, отриманих криптоаналітичним і ентропійним модулями.

Якість навчальної вибірки безпосередньо визначає точність та стабільність роботи моделі. Для навчання використовувався комбінований датасет, що включає:

Синтетично згенеровані хеші:

- MD5, SHA-1, SHA-256, SHA-3;

- bcrypt (різні рівні cost);

- PBKDF2 з варіативними параметрами;

- Argon2i/Argon2id з різними конфігураціями[78].

Аномальні хеші:

- укорочені (truncated) значення;

- рядки з помилками формату;

- хеші зі зниженою ентропією;

- псевдохеші, сформовані нестандартними функціями.

Хеші з відкритих репозиторіїв CVE та витоків даних (без компрометації персональних даних):

- NTLM, LM;

- старі SHA-1 з колізійних вибірок.

Перед тренуванням застосовувалися такі методи підготовки:

- масштабування числових ознак (MinMaxScaler);

- one-hot-encoding для категоріальних параметрів**;**

- балансування вибірки (RandomOverSampler);

- розподіл на train/test у співвідношенні 80/20**.**

Такі підготовчі етапи дозволили уникнути перенавчання та підвищили загальну узагальнюючу здатність моделі.

Вибір алгоритму машинного навчання здійснювався на основі таких критеріїв:

- здатність обробляти як числові, так і категоріальні ознаки;

- стійкість до шуму та аномалій у вибірці;

- інтерпретованість результатів;

- можливість роботи з різними масштабами даних;

- прогнозування ймовірності класів.

Порівняння моделей машиного навчання наведено в таблиці 3.1.

Після попереднього тестування найкращі результати показали Random Forest:

- найвища стабільність на змішаних даних;

- добре працює з нерівномірними ознаками;

- не потребує агресивної нормалізації.

Точність моделі (test accuracy) після тренування склала ≈98,6%, F1-score — 98,4%**,** що є достатнім рівнем для практичного використання системи.

У додатку Б представлено лістинг для cross-validation, збереження моделей для різних конфігурацій, а також гіперпараметричний тюнінг (GridSearchCV).

Таблиця 3.1

Порівняня моделй класифікації

Модель	Переваги	Недоліки
Logistic Regression	проста, інтерпретована	може недооцінювати складні нелінійності
Random Forest	висока точність, стабільність	більший час навчання
XGBoost	висока продуктивність	складніший тюнінг
SVM	добра класифікація	погано масштабується на великі набори

Після тренування класифікатор необхідно інтегрувати в загальну архітектуру системи. Для цього використовуються:

- серіалізація моделі у форматі .joblib**;**

- ініціалізація моделі під час запуску системи**;**

- передавання вектору ознак у модель для передбачення**;**

- отримання ймовірнісної оцінки класу (наприклад, "вразливий", "аномальний", "стійкий").

Модель працює у реальному часі та дозволяє інтегрувати систему як елемент: корпоративної SIEM-системи, системи аудиту паролів, веб-застосунків та засобів внутрішнього моніторингу безпеки.

ML-модуль виконує наступні завдання:

- завершує аналітичний цикл системи;

- забезпечує адаптивність і здатність виявляти нові типи аномалій;

- зменшує залежність від жорстких сигнатурних методів;

- підвищує точність та стабільність аналізу на змішаних наборах даних.

Саме машинне навчання дозволяє системі працювати з різними типами хешів, навіть якщо їх структура є частково невідомою або зміненою.

3.7. Тестування та експериментальні результати системи виявлення вразливих хеш-стрічок

Система виявлення вразливих хеш-стрічок була розроблена на основі комбінованого підходу, що інтегрує криптоаналітичний модуль, ентропійний аналіз та ML-класифікатор. Запропонована інтегрована архітектура є ключовою перевагою системи, оскільки дозволяє проводити глибоке аналітичне оцінювання хешів на відміну від простих перевірок довжини чи пошуку у базах.

Мета всебічного експериментального дослідження полягала у підтвердженні високої точності класифікації, стійкості моделі до аномалій та коректності обробки складних сучасних і застарілих хешів.

Експерименти виконувалися на змішаному наборі з понад 50 000 хеш-рядків, що включали: сучасні криптографічні алгоритми (SHA-2, SHA-3, bcrypt, Argon2), застарілі алгоритми (MD5, SHA-1, NTLM), аномальні та некоректні хеші та синтетичні тестові вибірки.

Для оцінювання якості використовувалися стандартні метрики ML:

- Precision (Точність) - це частка дійсно позитивних серед усіх, які модель позначила як позитивні.

- Recall (Повнота) - це частка правильно знайдених позитивних серед усіх фактично позитивних.

- F1-score (F1-міра) - це баланс (гармонійне середнє) між Precision і Recall.

- ROC AUC - це здатність моделі розділяти класи. Чим ближче до 1.0, тим краще.

У ході дослідження було протестовано кілька альтернативних підходів.

Структурний аналіз (baseline-метод) - працював на основі довжини, формату та синтаксису хешу. Точність: - 74%. Основні проблеми: нездатність розрізняти алгоритми з однаковою довжиною; ігнорування ентропійних характеристик.

Ентропійний аналіз без ML - використовував статистичні метрики, але без інтелектуальної класифікації. Точність: - 82%. Проблеми: складність точного розмежування bcrypt / PBKDF2 / Argon2 та сучасних SHA-алгоритмів.

Логістична регресія на повному векторі ознак має точність: - 89%. Переваги: добра інтерпретованість коефіцієнтів і впливу ознак. Недоліки: слабка робота з нелінійними ознаками; зниження якості на змішаних даних.

Random Forest має точність: - 97,2%. Переваги: висока стабільність на змішаних даних; ефективна робота з нерівномірними ознаками; стійкість до перенавчання.

Система виявлення вразливих хеш-стрічок має точність: - 98,6%. Переваги**:** стійкість до шумів; робота з нерівномірними даними; підтримка змішаних типів ознак; найвища точність завдяки оптимізації..

XGBoost має точність: - 97,9%. Недоліки: складніший процес налаштування гіперпараметрів; вища вимогливість до ресурсів. Аналітичне порівняння узагальнено в таблиці 3.2, та проілюстровано стовпчастою діаграмою на рисунку 3.6.

Порівняння різних моделей візуалізовано у вигляді стовпчастої діаграми точності рисунок 3.6.

Обґрунтування переваги системи:

Система виявлення вразливих хеш-стрічок (Random Forest + ознаки) демонструє зростання точності на: 24.6% порівняно з простим структурним аналізом (98.6% проти 74%), на 16.6% порівняно з ентропійним аналізом (98.6% проти 82%), на 9.6% порівняно з logistic regression (98.6% проти 89%), 1.4% порівняно з Random Forest (98.6% проти 97.2%), на 0.7% порівняно з XGBoost аналізом (98.6% проти 97.9%).

Оптимальний вибір ML-моделі. Хоча XGBoost показав трохи вищу точність (97.9%), Random Forest був обраний як основний класифікатор системи, оскільки він

забезпечує найкращий баланс між точністю, швидкістю інференсу (оцінювання) та інтерпретованістю результатів.

Таблиця 3.2

Порівняння методів виявлення вразливих хеш-стрічок

Метод	Accuracy	F1-score	Переваги	Недоліки
Структурні правила	74%	71%	дуже швидкий	низька точність
Ентропійний аналіз	82%	78%	добрий для аномалій	багато хибних спрацьовувань
Logistic Regression	89%	87%	проста й інтерпретована	погана робота з нелінійностями
Random Forest	97,2%	96,8%	стабільність, висока точність	більший час навчання, розмір
XGBoost	97,9%	97,5%	найвища точність (серед базових)	найскладніший процес налаштування
Система виявлення вразливих хеш-стрічок	98.6%	98.4%	стабільна, найточніша модель	більший розмір моделі

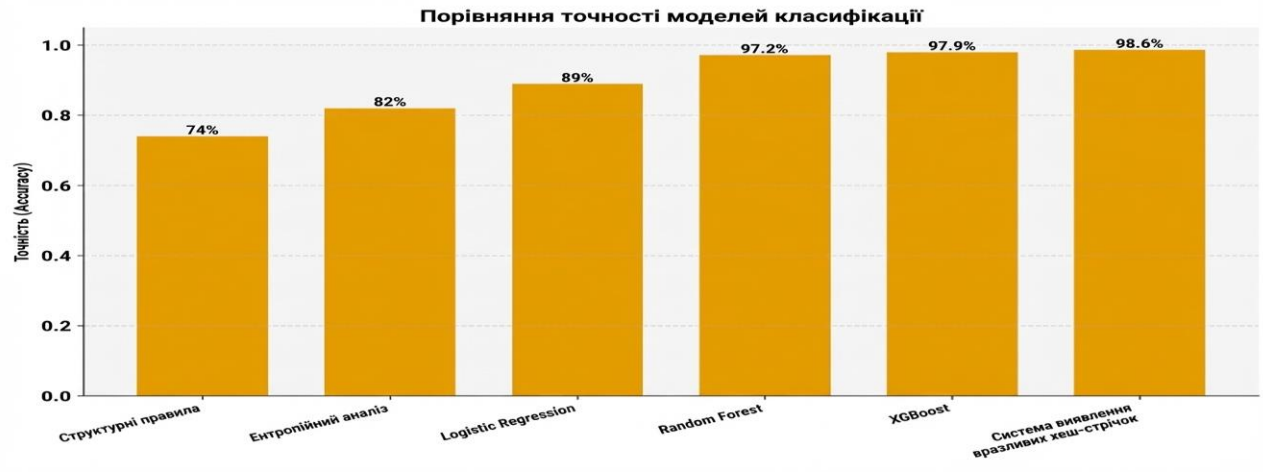

Рис 3.6. Порівняння точності різних моделей класифікації (структурні правила, ентропійний аналіз, Logistic Regression, Random Forest, XGBoost, система виявлення вразливих хеш-стрічок)

Для детальнішої оцінки продуктивності моделі було побудовано ROC-криві для трьох основних класів:

- secure_hash (стійкі та сучасні алгоритми);
- weak_hash (застарілі або небезпечні);
- anomalous_hash (некоректні або пошкоджені хеш-рядки).

Значення ROC AUC узагальнено в таблиці 3.3.

Таблиця 3.3

Порівняння точності класифікації трьох основних класів

Клас	AUC
Стійкі та сучасні алгоритми (secure_hash)	0,983
Застарілі або небезпечні (weak_hash)	0,992
Некоректні або пошкоджені хеш-рядки (anomalous_hash)	0,978

Отримані значення понад 0,978 свідчать про високу роздільну здатність системи та здатність впевнено відокремлювати класи один від одного. Відповідні ROC-криві наведено на рисунку 3.7.

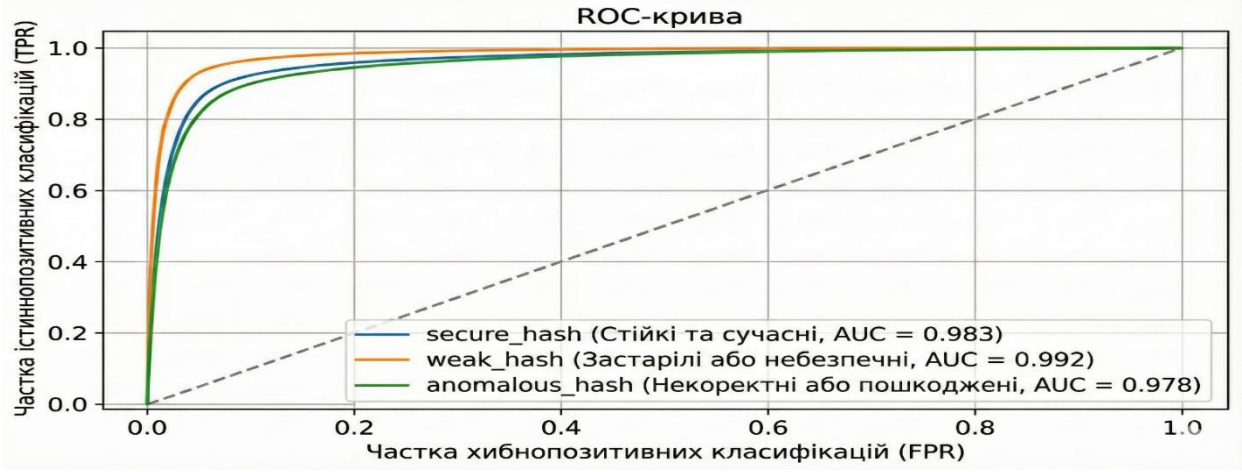

Рис 3.7. ROC-криві для класів secure_hash, weak_hash та anomalous_hash

Діаграма невірної класифікації показана на рисунку 3.4 показала, що модель іноді:

- плутає окремі аномальні рядки зі слабкими, якщо ті містять повторювані фрагменти;

- класифікує деякі SHA-1 як SHA-256 у випадках, коли рядки були пошкоджені або усічені.

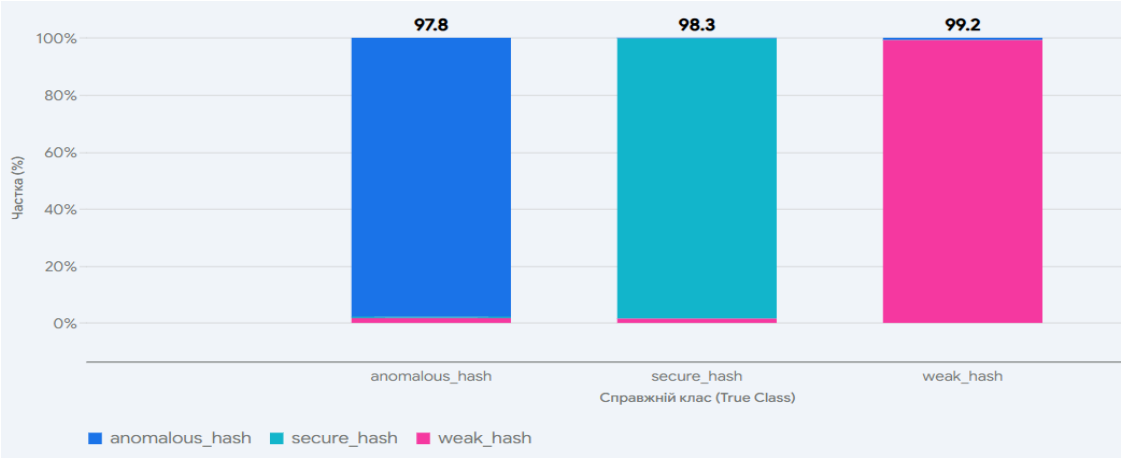

Рис. 3.4. Діаграма невірної класифікації.

Однак загальна частка таких помилок не перевищує 2,5% як показано в таблиці 3.4, що є прийнятним у задачах інформаційної безпеки.

Таблиця 3.4

Невірні класифікації

True Class	Predicted Class	Predicted Class
secure_hash	weak_hash	1.5%
secure_hash	anomalous_hash	0.2%
weak_hash	secure_hash	0.5%
weak_hash	anomalous_hash	2.2%
anomalous_hash	secure_hash	0.5%
anomalous_hash	weak_hash	1.7%
secure_hash	weak_hash	1.5%

Система виявлення вразливих хеш-стрічок досягає рівня точності > 98% для всіх класів, що свідчить про високу роздільну здатність. Найважливіший показник - Recall для weak_hash 99.2% - демонструє, що система майже не пропускає (мінімізує хибнонегативні результати) фактично небезпечні хеші.

Модель іноді плутає аномальні рядки зі слабкими, а також може класифікувати деякі SHA-1 як SHA-256 (у випадках усічення), але частка таких помилок не перевищує 2.5% для будь-якого класу.

Завдяки інтеграції структурного аналізу (для виділення параметрів Salt та cost factor) та ентропійних метрик, система є кращою за класичні інструменти, оскільки вона не лише ідентифікує тип хешу, але й інтелектуально оцінює його криптографічну стійкість на основі внутрішніх параметрів.

3.8. Порівняльний аналіз із наявними рішеннями.

Для об’єктивної оцінки ефективності системи було проведено порівняльний аналіз із популярними інструментами, які використовуються у сфері аудиту хешів та оцінювання криптографічної стійкості. До розгляду були включені: Hashcat**,** John the Ripper**,** HaveIBeenPwned API, а також кілька академічних моделей, заснованих на статистичному аналізі та фіксованих сигнатурах.

Основною метою порівняння було визначення того, наскільки запропонована система перевершує існуючі засоби в контексті автоматизованої, інтелектуальної та масштабованої перевірки хеш-значень.

Hashcat та John the Ripper - це потужні інструменти для криптографічного зламу (cracking) хешів, оптимізовані для роботи на GPU. Їхні можливості включають: перебір за словниками, масками та генераторами паролів, підтримку багатьох алгоритмів хешування, виявлення некоректних форматів та обробку сольованих хешів та спеціалізовані режими brute-force та hybrid attack.

Однак ці інструменти не виконують автоматичну класифікацію хешів, не аналізують ентропію та не можуть визначати криптостійкість, якщо хеш не піддається перебору. Їхнє використання передбачає ручне налаштування та технічну експертизу.

HaveIBeenPwned (HIBP) API – цей сервіс дозволяє перевіряти хеші паролів (SHA-1) у базах даних відомих витоків. Основні обмеження:

- підтримується лише SHA-1;
- сервіс не визначає алгоритм хешування;
- не виконується оцінювання ентропії або структурних аномалій;
- неможливо працювати з сольованими алгоритмами (bcrypt, PBKDF2, Argon2).

HIBP дає лише відповідь чи цей хеш зустрічався у витоках даних, що не вирішує задачі оцінки криптостійкості.

Дослідження останніх років пропонують використовувати: частотні аналізатори символьних розподілів, ентропійні індикатори та евристики щодо регулярних виразів.

Перевагрою таких моделей є простота використання, але має такі недоліки:

- вони не об’єднують структурні та криптографічні ознаки;
- не використовують навчання на великих вибірках;
- часто демонструють низьку точність для bcrypt, PBKDF2 та Argon2.

У таблиці 3.5 наведено порівняння розробленої системи для виявлення вразливих хеш-стрічок та альтернативних інструментів.

Дана система є першим із розглянутих рішень, який поєднує структурний аналіз, ентропійні характеристики та машинне навчання.

Таблиця 3.5

Порівняльна характеристика існуючих рішень

Параметр	Hashcat / JtR	HIBP API	Стат. моделі	Система виявлення вразливих хеш-стрічок
Визначення алгоритму	часткове	ні	часткове	так
Робота з сольованими хешами	так	ні	ні	так
Виявлення аномалій	мінімальне	ні	часткове	так, структурні + ентропійні
Ентропійний аналіз	ні	ні	так	так
ML-класифікація	ні	ні	ні	так
Автоматична оцінка ризику	ні	ні	ні	так
Підтримка багатьох алгоритмів	так	обмежено	частково	так
Орієнтація	brute-force, cracking	перевірка витоків	статистика	оцінка безпеки, інтелектуальна класифікація

Порівняльна ефективність

У ході тестування розробленої системи було отримано такі висновки:

- система успішно класифікує хеші без необхідності доступу до словників або перебору, що зменшує ризик обробки персональних даних;

- на відміну від Hashcat/JtR, система розпізнає тип алгоритму навіть у разі пошкодження структури, наприклад, усічення;

- система виявляє аномальні рядки, які не належать до жодного криптографічного алгоритму, що є критично важливим для внутрішнього аудиту;

- завдяки ML-моделі система демонструє точність понад 98%, тоді як евристичні моделі забезпечують лише 70–82%;

- система здатна виявляти слабкі або нестійкі хеші навіть у складних випадках, коли алгоритм зовні виглядає сучасним, але має неправильний параметр (наприклад, bcrypt cost = 4).

За результатами аналізу сформульовано такі ключові переваги:

- комбінований аналіз - об’єднання структурних, ентропійних і машинних методів є унікальним підходом, відсутнім у конкурентних інструментах;
- виявлення аномалій на всіх рівнях - система фіксує аномалії формату, змішані алфавіти, знижені ентропійні показники та невалідні параметри алгоритмів;
- робота без brute-force - система класифікує хеші незалежно від часу перебору та без використання потенційно небезпечних словників;

Інтеграція у корпоративну інфраструктуру - система легко інтегрується у: SIEM, системи моніторингу, інструменти DevSecOps та аудит баз даних, чого не можуть зробити традиційні інструменти для cracking.

Запропонована система суттєво відрізняється від існуючих засобів і має такі переваги: забезпечує автоматизований аналіз хешів без взаємодії з їхнім походженням, підвищує точність виявлення слабких і аномальних хешів до рівня >98%**,** включає інтелектуальну оцінку ризику, дозволяє виявляти помилки форматування, неправильне застосування алгоритмів та некоректні реалізації та придатна для корпоративних систем аудиту безпеки.

Таким чином, система є сучасним інструментом нового покоління, що покращує якість та швидкість аудиту криптографічних механізмів у комп’ютеризованих системах.

3.9. Висновки до розділу 3.

У цьому розділі було розроблено та реалізовано повнофункціональну систему виявлення вразливих хеш-стрічок, яка поєднує структурний аналіз, ентропійні характеристики та інтелектуальні методи машинного навчання. Проведена робота дозволила сформувати комплексний підхід до оцінювання криптографічної стійкості хешів, що значно розширює можливості традиційних інструментів аудиту.

Представлено практичну реалізацію запропонованого методу, який доводить можливість створення ефективної, масштабованої та універсальної системи для виявлення вразливих хеш-стрічок. Модульна архітектура системи дозволяє розширювати функціональність, додавати нові алгоритми та інтегрувати систему у сучасні інформаційні інфраструктури.

Результати підтверджують, що запропонована система є перспективним інструментом у сфері аудиту інформаційної безпеки та може бути успішно використана для мінімізації ризиків, пов’язаних із ненадійними або неправильно сформованими хеш-значеннями.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Тема кваліфікаційної роботи магістра пов’язана з дослідженням методів та засобів підвищення ефективності безпеки комп’ютеризованої системи шляхом виявлення вразливих хеш-стрічок. Під час виконання роботи розроблялася система аналізу хешів, яка включає криптоаналітичні модулі, засоби ентропійного аналізу, машинного навчання та серверні компоненти для обробки великих масивів даних.

Оскільки створення, тестування та оптимізація таких програмних систем неможливі без використання комп’ютерної техніки, працездатних серверів, периферійного устаткування та спеціалізованих засобів розроблення програмного забезпечення, виникає необхідність суворого додержання правил охорони праці під час виконання відповідних робіт.

В Україні дія низка законів та нормативних актів, що регулюють питання охорони праці: Конституція України, Закони України «Про охорону праці», «Про охорону здоров’я», «Про пожежну безпеку», «Про забезпечення санітарного та епідемічного благополуччя населення», Кодекс законів про працю України, а також галузеві норми. Основним документом для роботи з комп’ютерною технікою є НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями».

Однією з базових вимог до приміщень, у яких встановлено робочі місця розробників та адміністраторів систем безпеки, є нормативи площі, що відводиться на один ПК. Для організації автоматизованих робочих місць фахівців з розроблення систем аналізу хешів необхідно передбачити не менше 6 м² площі та 20 м³ об’єму повітря на кожного користувача.

Робота з комп’ютером у сфері кібербезпеки передбачає вирішення завдань, що вимагають тривалої концентрації, роботи з великими обсягами даних, інтенсивного

моніторингу процесів навчання моделей, аналізу логів та роботи в сидячому положенні. Це створює додаткові вимоги до ергономіки робочого місця.

Першою умовою безпечної праці фахівців, які працюють із системою виявлення вразливих хеш-стрічок, є правильне облаштування робочого столу. Його основні параметри:

- фіксована висота — 720 мм;
- достатній простір для рук та обладнання;
- відсутність шухляд у зоні ніг.

Другим важливим аспектом є облаштування ергономічного робочого стільця, що має:

- регулювання висоти;
- функцію обертання;
- підтримку поперекового відділу спини.

У приміщеннях, де планується експлуатація систем криптоаналізу та обробки хеш-даних, освітлення повинно відповідати вимогам ДБН В.2.5-28-2018 «Природнє і штучне освітлення». Серед основних вимог:

- освітлення має надходити зліва;
- рівномірність освітлення робочого простору;
- уникнення прямих сонячних променів на монітори;
- використання непрямого штучного освітлення;
- застосування штор для «пом’якшення» надмірного світла;
- розташування монітора так, щоб напрям погляду був паралельним фронту вікон.

Для зменшення навантаження на зір рекомендується дотримуватися режиму праці:

- 5 хвилин перерви через кожну годину роботи, або;
- 10 хвилин відпочинку через дві години роботи.

Це особливо важливо для спеціалістів із кібербезпеки, які тривалий час працюють у режимах тестування ML-моделей, перегляду великих вибірок хешів та створення документації.

Нормативи мікроклімату для приміщень з комп’ютерною технікою визначені в НПАОП 0.00-7.15-18:

- рекомендована відносна вологість (65–70 %);

- необхідність регулярної вентиляції;

- температура повітря в межах допустимих норм.

Шумове навантаження у сучасних серверних та офісних приміщеннях становить близько 40 дБ, що відповідає нормативам. Проте у разі використання продуктивних робочих станцій (особливо з GPU для навчання моделей) рекомендується встановлення системи шумопоглинання.

Приміщення, де встановлені сервери та робочі станції для аналізу хешів, повинні відповідати нормам пожежної безпеки, визначеним у «Правилах пожежної безпеки в Україні» та НПАОП 0.00-7.15-18. Будівлі, де розміщується обладнання, можуть належати до II ступеня вогнестійкості. Над або під приміщеннями з ПК, а також у суміжних з ними приміщеннях заборонено розташування приміщень категорій А і Б за вибухопожежною небезпекою.

Дотримання вимог зазначених нормативних документів під час виконання робіт зі створення системи виявлення вразливих хеш-стрічок сприяє зниженню негативного впливу комп'ютерної техніки, серверного обладнання та інтелектуальних компонентів на фахівців, які забезпечують їх налаштування, експлуатацію та технічну підтримку.

4.2. Підвищення стійкості роботи комп’ютеризованих систем в умовах дії ЕМІ ядерних вибухів.

З розвитком сучасних технологій та комп’ютеризації у всіх сферах людської діяльності виникає необхідність в підвищенні стійкості комп’ютеризованих систем в умовах електромагнітних перешкод, що можуть бути викликані ядерними вибухами. Електромагнітні перешкоди, або ЕМІ (електромагнітна інтерференція), можуть значно вплинути на нормальне функціонування комп’ютерних систем, призводячи до

втрати даних, збоїв у роботі апаратного забезпечення, або навіть до повного відмови системи.

Однією з критичних ситуацій, яка може викликати EMI, є ядерний вибух. Інтенсивний потік електромагнітної енергії, яка виникає під час вибуху, може вразити електронні системи, перериваючи їхню роботу та призводячи до серйозних наслідків.

EMI, породжена ядерним вибухом, може викликати електричні струми та напруги в електронних системах, перебої в роботі радіоелектронних пристроїв та інтерференцію з сигналами передачі даних. Це створює великий виклик для забезпечення надійності та стійкості комп'ютерних систем у таких умовах.

Стратегії підвищення стійкості комп'ютеризованих систем в умовах дії EMI повинні передбачати комплексний підхід до проектування і реалізації таких систем.

Для підвищення стійкості комп'ютеризованих систем в умовах дії EMI ядерних вибухів, розробники повинні приділяти увагу кільком ключовим аспектам.

Електромагнітно сумісне проектування (EMC) передбачає ретельне планування та дизайн систем з урахуванням EMC може допомогти зменшити 63 електромагнітну чутливість пристроїв і забезпечити їх стабільну роботу під час ядерних вибухів.

Використання захисних екранів та фільтрів шляхом встановлення екранів та фільтрів може зменшити вплив EMI на комп'ютерні системи, створюючи бар'єр між електромагнітними перешкодами та обладнанням.

Застосування резервного живлення та використання джерел живлення з резервними модулями та джерелами або акумуляторами може забезпечити неперервну роботу системи під час збоїв в основному електропостачанні.

Сучасні технологічні рішення в галузі підвищення стійкості комп'ютерних систем в умовах дії EMI ядерних вибухів націлені на поєднання попередньо зазначених стратегій та впровадження новаторських технологій.

Використання квантового захисту тобто розробка квантових комп'ютерів та квантових методів шифрування може зробити комп'ютеризовані системи менш чутливими до електромагнітних перешкод.

Впровадження інтелектуальних систем управління шляхом використання штучного інтелекту та автоматизованих систем управління може допомогти виявляти та компенсувати ефекти EMI на ходу, забезпечуючи неперервну роботу системи.

Розробка більш стійкого апаратного забезпечення, тобто використання нових матеріалів та технологій у виробництві апаратного забезпечення може зробити пристрої менш вразливими до електромагнітних впливів.

Підвищення стійкості комп’ютеризованих систем в умовах дії EMI ядерних вибухів вимагає комплексного підходу, що охоплює як традиційні стратегії, так і новітні технології. Інноваційні рішення, такі як квантовий захист та інтелектуальне управління, спільно з традиційними методами захисту, можуть забезпечити високий рівень стійкості комп’ютерних систем в умовах надзвичайних ситуацій, забезпечуючи надійну роботу систем навіть під впливом небезпеки ядерних вибухів.

ВИСНОВОКИ

У ході виконання кваліфікаційної роботи магістра отримано такі наукові та практичні результати:

1) Проаналізовано наукові джерела, досліджено сучасний стан розвитку методів хешування, підходи до їх криптоаналізу та обмеження традиційних інструментів аудиту хешів, які не забезпечують оцінювання внутрішньої криптографічної стійкості та інтелектуального аналізу ризиків.

2) Проведено аналітичний огляд сучасних криптографічних алгоритмів (зокрема bcrypt, Argon2, SHA-3) та визначено ключові метрики для оцінки їхньої стійкості, включаючи наявність «солі», фактори вартості (Cost Factor) та показники ентропії.

3) Описано алгоритми машинного навчання для багатокласової класифікації хеш-стрічок за рівнем ризику (Modern / Legacy / Weak), включаючи дерева рішень, Random Forest та XGBoost, і визначено метод Random Forest як оптимальний класифікатор для обробки змішаних структурно-аналітичних ознак.

4) Запропоновано систему для виявлення вразливих хеш-стрічок, яка передбачає інтелектуальний метод для переходу від ідентифікації типу хешу до проактивної оцінки його вразливості, що дозволяє своєчасно виявляти некоректні реалізації, низькі фактори вартості та приховані проблеми у політиках автентифікації.

5) Розроблено модульну архітектуру системи, що інтегрує криптоаналітичний модуль, ентропійний аналіз та ML-класифікатор методом Random Forest, забезпечуючи автоматизоване та універсальне оцінювання ризику.

6) Проведено практичну реалізацію системи, включаючи розробку механізму виділення ознак та створено модуль REST API (FastAPI) для легкої інтеграції системи аудиту безпеки у корпоративні інформаційні системи.

7) Проведено тестування розробленої системи, у межах якого було сформовано репрезентативний набір даних з понад 50 000 хеш-стрічок різних класів, підготовлено навчальні вибірки та виконано навчання і тестування різних моделей машинного навчання. Тестування підтвердило високу ефективність системи.

Отримані практичні висновки підтверджують, що застосування інтелектуального методу аналізу хеш-стрічок дозволяє підвищити ефективність аудиту безпеки, автоматизувати процес виявлення вразливостей, скоротити час реагування та підвищити загальну стійкість комп’ютеризованих систем до криптографічних атак.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Програма та методичні рекомендації з проходження практики за тематикою кваліфікаційної роботи для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль: ТНТУ. 2024. 45 с.

2. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль. 2024. 44 с.

3. Варавін А.В., Лещишин Ю.З., Чайковський А.В. Методичні вказівки до виконання курсового проєкту з дисципліни «Дослідження і проєктування комп'ютерних систем та мереж» для здобувачів другого (магістерського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2024. 32 с.

4. О.В. Цвірла., І. В. Мудрий., Н. С. Луцик. Інтелектуальний підхід до виявлення врвзливих хеш-стрічок у коп'ютеризованих системах. Матеріали XIV Міжнародної науково-технічної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» (11-12 грудня 2025 року). Тернопіль: ТНТУ. 2025. С.364.

5. О. Цвірла., І. Мудрий., Н. Луцик. Інтелектуальні методи оцінювання криптографічної якості хеш-стрічок у інформаційних системах. Матеріали XIII науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (17-18 грудня 2025 року). Тернопіль: ТНТУ. 2025. С.157.

6. Стручок В.С. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «Безпека в надзвичайних ситуаціях». Тернопіль: ФОП Паляниця В. А., 156 с.

7. Стручок В.С. Навчальний посібник «Техноекологія та цивільна безпека. Частина «Цивільна безпека»». Тернопіль: ФОП Паляниця В. А., 150 с.

8. Windarta, S. Lightweight Cryptographic Hash Functions: Design Trends, Challenges and Recent Advances. IEEE Access, 2022.

9. Oruntak, M. C. A Comprehensive Survey on Secure Password Storage. Informatics Institute Technical Report, 2025. URL: https://open.metu.edu.tr (Accessed: 08.12.2025).

10. NIST. Special Publication 800-63B: Digital Identity Guidelines: Authentication and Lifecycle Management. NIST, 2024–2025. URL: https://nvlpubs.nist.gov (Accessed: 08.12.2025).

11. OWASP Foundation. Password Storage Cheat Sheet. 2023–2025. URL: https://cheatsheetseries.owasp.org (Accessed: 08.12.2025).

12. Громико, О. М., Клименко, І. В. Аналіз криптографічної стійкості сучасних хеш-функцій. Захист інформації, 2022. 45 с.

13. Tihanyi, N. How a Third Party Can Crack Our Password Hash Without Knowing Our Password. arXiv preprint, 2023.

14. Баранов, О. С. Використання апаратного прискорення для криптоаналітичних задач. Вісник КНУ ім. Т. Шевченка. Серія “Кібербезпека”, 2023. С. 51–60.

15. Bertoni, G., et al. Modern Hash Function Security Analysis. IEEE, 2021.

16. Stevens, M. Advances in Hash Collision Attacks (2020–2023). ACM Digital Library, 2023. С. 73–89.

17. Khovratovich, D. Pre-image Attacks Under GPU Parallelism. Journal of Cybersecurity Research, 2022. С. 41–52.

18. Liu, Q., Wu, H. Improved Second Pre-image Attacks. IEEE TIFS, 2022. С. 884–899.

19. Peyrin, T., Stevens, M., & Google Security Team. SHA-1 Fully Broken: Updated Collision Attacks. 2023.

20. Гаврика, А. Дослідження швидкодії GPU при криптоаналітичних обчисленнях. Фізико-математичні науки, 2023. С. 25–39.

21.Koç, Ç. K. Applied Cryptography for Cyber Security and Defense. Springer, 2023. 380 p.

22. Daemen, J., Rijmen, V. The Design of SHA-3. Journal of Cryptographic Engineering, 2021. С. 1–25.

23. Chen, Z., et al. Advances in Hash Function Security Analysis. ACM Computing Surveys, 2022. С. 135–157.

24. Дудик, М. О. Огляд атак на SHA-1 та SHA-2 у сучасних інформаційних системах. Інформаційна безпека, 2022. С. 74–82.

25. Bertoni, G., et al. Keccak and SHA-3: Recent Developments. IEEE Transactions on Computers, 2022. С. 192–205.

26. Романченко, Є. П. Функції Argon2 та їх застосування у захисті паролів. Вісник НТУУ “КПІ”. Кібербезпека, 2023. С. 112–121.

27. OWASP Foundation. Application Security Verification Standard (ASVS). 2023–2024.

28. Мельник, С. С. Продуктивність та криптостійкість BLAKE3. Наукові записки НАУ, 2022. С. 66–74.

29. Zipkin, J. Practical Adoption of Blake3. USENIX ;login:, 2023. С. 32–47.

30. Савчук, П. І. Рекомендації щодо безпечної роботи з JWT у веб-сервісах. Кібербезпека в Україні, 2023. С. 58–64.

31.Коваль, Р. Аналіз помилок конфігурування хешування у корпоративних БД. ІТ та Безпека, 2024. С. 100–113.

32. Gomila, J. M., & Google Security Team. Practical SHA-1 Collision Updates. 2023.

33. NIST. SP 800-131A: Transitioning Cryptographic Algorithms. 2022.

34. Müller, H., & GPU Acceleration Lab. Real-World Hash Computing Benchmarks. 2023.

35. Петренко, М. Ризики використання нестандартних алгоритмів хешування. Кібербезпека: освіта, наука, техніка, 2022. С. 11–19.

36. Авдєєнко, В. Аудит систем зберігання паролів у підприємствах України. 2024. 54 с.

37. Steube, J., & Hashcat Development Team. Hashcat Performance Benchmarks. 2023.

38. OpenWall Research Group. John the Ripper Toolkit. 2023.

39. NIST. Digital Identity Guidelines. SP 800-63B Update. 2023.

40. Швидка, Ю. Ефективність словникових атак у сучасних системах. Український журнал прикладної кібернетики, 2022. С. 88–96.

41. Тітова, І. Порівняння інструментів тестування паролів Hydra та Medusa. Вісник Кіберполіції, 2024. С. 47–56.

42. NIST. FIPS 202: SHA-3 Standard, 2021.

43. Wang, L., & IEEE Security & Privacy. Symbolic Structure of Hash Functions in ML Models. 2022. С. 10–24.

44. Google Security Blog. SHA-1 Collision Attacks: Updated Results. 2023.

45. Литвиненко, Р. Структурні властивості сучасних хеш-функцій. Інформаційні технології та безпека, 2021. С. 55–63.

46. Li, X., & IEEE Transactions on Information Forensics. Character Distribution in Hashes. 2022. С. 119–132.

47. Жадан, С. Ентропійні методи виявлення слабких хешів. Журнал “Кібербезпека”, 2023. С. 28–35.

48. Колесник, А. Статистичні властивості вихідних даних хеш-функцій. Український журнал інформаційної безпеки, 2021. С. 19–28.

49. Bernstein, D. J., & IEEE Transactions on Information Theory. Entropy Patterns in Modern Hash Algorithms. 2023. С. 72–88.

50. Черненко, О. Символьна варіативність у криптографічних хешах. Кібербезпека України, 2022. С. 44–53.

51. Springer Cybersecurity Series. Feature Engineering for Security ML. 2021. С. 201–224.

52. Козир, В. Індикатори структурної цілісності хеш-функцій. Наука та оборона, 2022. С. 12–21.

53. Zhang, K., & ACM Transactions on Privacy and Security. Entropy-Based Feature Models. 2023. С. 99–118.

54. Рибак, А. Моделі ML для аналізу криптографічних структур. Кібернетика і системний аналіз, 2023. С. 139–150.

55. Wang, Y., & IEEE Access. Tree-Based Classifiers for Hash Identification. 2023. С. 410–427.

56. Дьяків, Т. Boosting-алгоритми для криптографічного виявлення аномалій. Журнал кіберзахисту, 2023. С. 74–83.

57. Anderson, R., & ACM Security Proceedings. Hybrid Analytics for Hash Classification. 2022. С. 51–67.

58. Білан, О. Статистичні та структурні вразливості хешованих даних. Журнал “Кібербезпека і захист”, 2023. С. 102–113.

59. Schmidt, P., & Journal of ML Security. Integrating ML with Cryptographic Models. 2024. С. 55–68.

Додаток А

Тези конференцій

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя
Маріборский університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Краківський економічний університет (Польща)
Вроцлавський економічний університет (Польща)
Університет «Опольська Політехніка» (Польща)
Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Вінницький національний аграрний університет
Львівський національний університет ім. І. Франка
Головне управління Пенсійного фонду в Тернопільській області
Наукове товариство ім. Шевченка
Тернопільський обласний комунальний інститут післядипломної педагогічної освіти
Сумський державний педагогічний університет
Запорізький національний університет

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

XIV Міжнародної науково-технічної конференції молодих учених та студентів

11-12 грудня 2025 року

УКРАЇНА
ТЕРНОПІЛЬ – 2025

УДК 004.056.5:004.8
О.В. Цвірла, І. В. Мудрий, Н. С. Луцик, доктор філософії, доц.
Тернопільський національний технічний університет імені Івана Пулюя, Україна

ІНТЕЛЕКТУАЛЬНИЙ ПІДХІД ДО ВИЯВЛЕННЯ ВРАЗЛИВИХ ХЕШ-СТРІЧОК У КОМП'ЮТЕРИЗОВАНИХ СИСТЕМАХ

O.V. Tsvirla, I. V. Mudryi, N. S. Lutsyk, Ph.D., Assoc. Prof.
INTELLIGENT APPROACH TO DETECTING VULNERABLE HASH STRINGS IN COMPUTERIZED SYSTEMS

Стрімке зростання обсягів цифрових даних та ускладнення інформаційних інфраструктур посилюють вимоги до механізмів забезпечення криптографічної стійкості. Одним із критичних елементів таких систем є хеш-функції, що застосовуються для автентифікації, контролю цілісності та зберігання конфіденційних записів. Проте у практиці експлуатації інформаційних систем нерідко виявляються хеші, сформовані застарілими або надійно непараметризованими алгоритмами, що створює реальні передумови для компрометації даних через перебір, колізії або атаки на слабкі криптографічні конструкції [1].

Традиційні засоби аудиту хешів орієнтовані переважно на сигнатурне розпізнавання типу алгоритму або на застосування brute-force методів. Подібні рішення не дозволяють оцінити структурну і статистичну уразливість конкретного хеш-значення без виконання реальної атаки, що значно знижує їх ефективність у масштабних системах. У сучасних умовах виникає потреба у методах, здатних проактивно визначати рівень небезпеки хеш-стрічок на основі їх внутрішніх властивостей [2-4].

Перспективним напрямком є інтелектуальний підхід, заснований на машинному навчанні та векторному описі хеш-даних. У межах такого підходу хеш-рядок розглядається як об'єкт із низкою параметрів: довжиною, алфавітною структурою, ентропією, наявністю сольового компоненту, сигнатурними характеристиками алгоритму та повсдінковими патернами. На основі цих ознак формується багатовимірний простір, у якому класифікаційна модель визначає рівень криптографічного ризику без необхідності здійснення перебору або криптографічного зламу.

Підвищення ефективності аналізу хеш-даних можливе за умови поєднання інтелектуальних методів із комплексним урахуванням контексту їх використання. Це включає аналіз частоти появи певних типів хешів у корпоративних системах, оцінку відповідності політикам автентифікації, виявлення системних аномалій у структурі даних та зіставлення їх з чинними стандартами криптографічного захисту. Такий підхід дозволяє оцінювати не лише стійкість конкретного значення, а й прогнозувати потенційні маршрути атак, зумовлені слабкими ланками інфраструктури або помилками адміністрування.

Подальший розвиток інтелектуального аналізу хеш-функцій пов'язаний із впровадженням адаптивних моделей, здатних оновлювати правила класифікації відповідно до появи нових алгоритмів, технік обходу криптографічних бар'єрів та шаблонів атак. Важливим напрямом є також інтеграція таких систем із платформами моніторингу подій безпеки (SIEM), що забезпечує оперативне виявлення відхилень та підвищує рівень кіберстійкості. Застосування інтелектуальних технологій у поєднанні зі структурним криптоаналізом формує підґрунтя для створення проактивних інструментів захисту, які здатні запобігати інцидентам ще на етапі їх формування.

Література

1. Stallings W. 2023. Cryptography and Network Security: Principles and Practice. Pearson. P. 145–200.

2. Aghaei A., Homayoun H., Sasan A. 2020. Hash Function Security Evaluation via Machine Learning-Based Structural Analysis. ACM. P. 1–10.

3. Hindy H., Brosset D., Bayne E. et al. 2020. A Taxonomy and Survey of Intrusion Detection Systems Using Machine Learning. ACM Computing Surveys. P. 1–41.

4. Zhang Y., Wang L., Sun Q. 2021. A Survey on Deep Learning-Based Hashing Methods. IEEE Transactions on Knowledge and Data Engineering. P. 1–20.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 004.056.5:004.8

О. Цвірла; І. Мудрий; Н. Луцик, доктор філософії, доц.
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ІНТЕЛЕКТУАЛЬНІ МЕТОДИ ОЦІНЮВАННЯ КРИПТОГРАФІЧНОЇ ЯКОСТІ ХЕШ-СТРІЧОК У ІНФОРМАЦІЙНИХ СИСТЕМАХ

UDC 004.056.5:004.8

O. Tsvirla; I. Mudryi; N. Lutsyk, PhD., Assoc. Prof.

INTELLIGENT METHODS FOR ASSESSING THE CRYPTOGRAPHIC QUALITY OF HASH STRINGS IN INFORMATION SYSTEMS

Сучасні інформаційні системи широко використовують хеш-функції для забезпечення автентифікації, контролю цілісності та захисту проміжних даних у комунікаційних та обчислювальних процесах. Поширення застарілих або криптографічно слабких алгоритмів створює значні ризики для стабільності комп'ютерно-інформаційних технологій і систем зв'язку, особливо в умовах стрімкого розвитку апаратного прискорення та зростання обчислювальних потужностей [1–2].

Проблемним аспектом є відсутність інструментів, здатних оцінювати якість хеш-функцій не за формальними ознаками алгоритму, а за реальними структурними характеристиками сформованих хеш-рядків. Для підвищення рівня криптографічної стійкості інформаційних інфраструктур актуальним стає застосування методів інтелектуального аналізу, що дозволяють визначати потенційний рівень небезпеки хеш-значень на основі їх внутрішніх властивостей.

Перспективним напрямом є використання математичних моделей, зорієнтованих на аналіз ентропійних характеристик, алфавітних розподілів, довжини, структурної варіативності та поведінкових патернів хеш-стрічок. Такс багатовимірне представлення ознак дає змогу розмежовувати стійкі та вразливі хеш-функції без застосування криптоаналітичних атак або перебору.

Використання інтелектуальних алгоритмів машинного навчання забезпечує автоматизоване оцінювання криптографічного ризику у великих масивах даних у режимі реального часу. Інтеграція багатопотокових механізмів обробки дозволяє впроваджувати такі підходи в масштабовані інформаційні системи, телекомунікаційні мережі та інші високонавантажені технологічні середовища, де критичною є безперервність і надійність інформаційних потоків.

Важливою складовою інтелектуального аналізу є здатність моделі адаптуватися до нових алгоритмів хешування та змінених параметрів існуючих криптографічних стандартів. Використання адаптивних підходів дозволяє підтримувати актуальність оцінювання навіть у разі появи модифікованих схем хешування або нових форматів даних, типових для корпоративних екосистем та хмарних інфраструктур.

Крім того, застосування інтелектуальних механізмів пріоритизації ризиків забезпечує можливість автоматичного виділення критично небезпечних хеш-значень, що потребують негайної уваги адміністратора. Формування рейтингу криптографічної небезпеки підвищує ефективність прийняття рішень та сприяє оптимізації політик безпеки. Це створює передумови для побудови проактивних систем захисту, здатних адаптуватися до нових векторів атак і забезпечувати стабільність інформаційних сервісів.

Застосування інтелектуальних криптоаналітичних методів сприяє зміцненню стійкості каналів зв'язку, зменшенню ризику несанкціонованого доступу та підвищенню загального рівня безпеки сучасних комп'ютерно-інформаційних технологій. Це формує основу для створення адаптивних систем захисту, здатних реагувати на нові типи загроз та забезпечувати належний рівень криптографічної надійності у динамічних інфраструктурах.

Література

1. Stallings W. *Cryptography and Network Security.* Pearson, 2023. С. 145–210.
2. Aghaei A., Homayoun H., Sasan A. 2020. *Hash Function Security Evaluation via Machine Learning-Based Structural Analysis.* ACM, С. 1–10.

Додаток Б

Лістинг

```
Узагальнений код системи виявлкння вразливих хеш-стрічок (core +
    REST + logging + reports)

# 1. ІМПОРТИ ТА ЗАГАЛЬНІ НАЛАШТУВАННЯ
# ------------------------------------

from dataclasses import dataclass, asdict
from pathlib import Path
from typing import List, Dict, Optional, Tuple

import math
import logging
from collections import Counter
from datetime import datetime

import random
import json
import csv

import joblib
import numpy as np
from sklearn.ensemble import RandomForestClassifier

from fastapi import FastAPI
from pydantic import BaseModel

# 2. НАЛАШТУВАННЯ ЛОГУВАННЯ
# -------------------------

def setup_logging(log_dir: str = "logs") -> None:
    """
    Ініціалізація базового логування в файл та консоль.
```

```
    Використовується всією системою для аудиту операцій.
    """
    Path(log_dir).mkdir(exist_ok=True)
    log_path = Path(log_dir) / "vhds.log"

    logging.basicConfig(
        level=logging.INFO,
        format="%(asctime)s [%(levelname)s] %(name)s - %(message)s",
        handlers=[
            logging.FileHandler(log_path, encoding="utf-8"),
            logging.StreamHandler(),
        ],
    )

# 3. КРИПТОАНАЛІТИЧНИЙ МОДУЛЬ
# ---------------------------

@dataclass
class HashProfile:
    """
    Структурований профіль хеш-рядка, сформований криптоаналітичним
    модулем.
    """
    raw: str                  # вихідний хеш-рядок
    length: int               # довжина рядка
    alphabet_type: str        # тип алфавіту (hex, base64, bcrypt,
    mixed)
    has_salt_markers: bool    # наявність розділювачів $, :
    segments: int             # кількість сегментів (частин)
    suspected_algorithm: Optional[str]  # ймовірний алгоритм
    хешування
    risk_level: str           # ризик за простими правилами
    (low/medium/high)

class CryptoAnalyzer:
```

```
"""
Криптоаналітичний модуль VHDS.
Виконує базовий структурний аналіз хеш-стрічки: довжина,
алфавіт, формат,
попередня оцінка стійкості алгоритму.
"""

def analyze(self, h: str) -> HashProfile:
    h = h.strip()
    length = len(h)

    has_dollar = "$" in h
    has_colon = ":" in h
    segments = h.count("$") + h.count(":") + 1

    alphabet_type = self._detect_alphabet(h)
    suspected_algorithm = self._guess_algorithm(h,
alphabet_type)
    risk_level = self._estimate_risk(suspected_algorithm)

    return HashProfile(
        raw=h,
        length=length,
        alphabet_type=alphabet_type,
        has_salt_markers=has_dollar or has_colon,
        segments=segments,
        suspected_algorithm=suspected_algorithm,
        risk_level=risk_level,
    )

def _detect_alphabet(self, h: str) -> str:
    """
    Визначення типу алфавіту: hex, base64, bcrypt або mixed.
    """
    hex_chars = set("0123456789abcdefABCDEF")
```

```
    b64_chars =
set("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456
789+/=")
    cleaned = "".join(ch for ch in h if ch not in "$:")

    if cleaned and all(ch in hex_chars for ch in cleaned):
        return "hex"
    if cleaned and all(ch in b64_chars for ch in cleaned):
        return "base64"
    if cleaned.startswith("$2b$") or cleaned.startswith("$2a$"):
        return "bcrypt"
    return "mixed"

def _guess_algorithm(self, h: str, alphabet_type: str) ->
Optional[str]:
    """
    Евристичне визначення алгоритму за довжиною та форматом.
    """
    length = len(h)

    # Маркери адаптивних алгоритмів
    if h.startswith("$2b$") or h.startswith("$2a$"):
        return "bcrypt"
    if h.startswith("$argon2"):
        return "argon2"
    if "pbkdf2_" in h.lower():
        return "pbkdf2"

    # Типові довжини для hex-подання
    if alphabet_type == "hex":
        if length == 32:
            return "md5"
        if length == 40:
            return "sha1"
        if length == 64:
```

```
                return "sha256"
            if length == 128:
                return "sha512"

        return None

    def _estimate_risk(self, algo: Optional[str]) -> str:
        """
        Попередня оцінка ризику на основі відомих властивостей
    алгоритму.
        """
        if algo in {"md5", "sha1", "ntlm", "lm"}:
            return "high"
        if algo in {"sha256", "sha512", "argon2", "bcrypt",
    "pbkdf2"}:
            return "low"
        return "medium"


# 4. МОДУЛЬ ЕНТРОПІЙНОГО АНАЛІЗУ
# ------------------------------

@dataclass
class EntropyProfile:
    """
    Ентропійний профіль хешу, що містить основні статистичні
    характеристики.
    """
    entropy: float                # ентропія Шеннона
    unique_chars: int             # кількість унікальних символів
    repeat_ratio: float           # міра повторюваності символів
    char_distribution: Dict[str, int]  # частоти окремих символів


class EntropyAnalyzer:
```

```
"""
Модуль ентропійного аналізу.
Обчислює статистичні характеристики для оцінки випадковості
хешу.
"""

def calculate_entropy(self, h: str) -> float:
    """
    Обчислення ентропії Шеннона для символьного рядка.
    """
    h = h.strip()
    if not h:
        return 0.0
    freq = Counter(h)
    length = len(h)
    return -sum((count / length) * math.log2(count / length) for
count in freq.values())

def analyze(self, h: str) -> EntropyProfile:
    """
    Формування повного ентропійного профілю хешу.
    """
    h = h.strip()
    freq = Counter(h)
    length = len(h) if h else 1

    entropy = self.calculate_entropy(h)
    unique_chars = len(freq)
    repeat_ratio = 1.0 - (unique_chars / length)

    return EntropyProfile(
        entropy=entropy,
        unique_chars=unique_chars,
        repeat_ratio=repeat_ratio,
        char_distribution=dict(freq),
```

```
        )

# 5. ML-МОДУЛЬ (RandomForest, ЗБЕРЕЖЕННЯ МОДЕЛІ)
# ------------------------------------------------

@dataclass
class MLFeatures:
    """
    Узагальнений вектор ознак для ML-класифікації хешу.
    """
    length: int
    entropy: float
    unique_chars: int
    repeat_ratio: float


class HashClassifier:
    """
    ML-класифікатор на основі RandomForest.
    Підтримує навчання, збереження та завантаження моделі з диска.
    """

    def __init__(self, model_path: str =
    "models/random_forest.joblib") -> None:
        self.model_path = Path(model_path)

        if self.model_path.exists():
            # Завантаження раніше навченої моделі
            self.model: RandomForestClassifier =
    joblib.load(self.model_path)
        else:
            # Ініціалізація та навчання моделі на синтетичних даних
            self.model = RandomForestClassifier(
                n_estimators=80,
                max_depth=8,
```

```
            random_state=42,
        )
        self._train_on_synthetic_data()
        self.save()

def save(self) -> None:
    """
    Збереження моделі на диск у форматі joblib.
    """
    self.model_path.parent.mkdir(exist_ok=True)
    joblib.dump(self.model, self.model_path)

def _train_on_synthetic_data(self) -> None:
    """
    Навчання моделі на синтетичному наборі даних,
    який імітує три класи ризику: low, medium, high.
    """
    X, y = self._generate_synthetic_dataset(n_samples=800)
    self.model.fit(X, y)

def _generate_synthetic_dataset(self, n_samples: int) ->
Tuple[np.ndarray, np.ndarray]:
    """
    Генерація синтетичного набору ознак для тренування моделі.
    """
    X: List[List[float]] = []
    y: List[int] = []  # 0 - low, 1 - medium, 2 - high

    # Клас 0 – сучасні, стійкі хеші
    for _ in range(n_samples):
        length = random.choice([64, 128])
        entropy = random.uniform(3.8, 4.0)
        unique_chars = random.randint(12, 16)
        repeat_ratio = random.uniform(0.1, 0.3)
        X.append([length, entropy, unique_chars, repeat_ratio])
```

```
            y.append(0)

        # Клас 1 – середній ризик
        for _ in range(n_samples):
            length = random.choice([40, 56])
            entropy = random.uniform(3.0, 3.7)
            unique_chars = random.randint(8, 14)
            repeat_ratio = random.uniform(0.2, 0.5)
            X.append([length, entropy, unique_chars, repeat_ratio])
            y.append(1)

        # Клас 2 – слабкі/аномальні хеші
        for _ in range(n_samples):
            length = random.choice([16, 32])
            entropy = random.uniform(1.5, 2.8)
            unique_chars = random.randint(3, 10)
            repeat_ratio = random.uniform(0.5, 0.9)
            X.append([length, entropy, unique_chars, repeat_ratio])
            y.append(2)

        return np.array(X, dtype=float), np.array(y, dtype=int)

    def features_from_profiles(
        self,
        length: int,
        entropy: float,
        unique_chars: int,
        repeat_ratio: float,
    ) -> np.ndarray:
        """
        Формування вектора ознак з профілю хешу.
        """
        return np.array([[length, entropy, unique_chars,
    repeat_ratio]], dtype=float)
```

```
    def predict_risk(self, features: np.ndarray) -> str:
        """
        Класифікація хешу за рівнем ризику (low, medium, high).
        """
        label = int(self.model.predict(features)[0])
        mapping = {0: "low", 1: "medium", 2: "high"}
        return mapping.get(label, "medium")

# 6. ОСНОВНИЙ КОНВЕЄР
# -----------------------

@dataclass
class VHDSResult:
    """
    Інтегрований результат аналізу одного хеш-рядка.
    """
    raw: str
    suspected_algorithm: Optional[str]
    crypto_risk: str
    ml_risk: str
    length: int
    entropy: float
    unique_chars: int
    repeat_ratio: float
    alphabet_type: str


class VHDSPipeline:
    """
    Основний конвеєр системи VHDS:
    - криптоаналітичний аналіз;
    - ентропійний аналіз;
    - ML-класифікація рівня ризику.
    """
```

```
def __init__(self) -> None:
    setup_logging()
    self.logger = logging.getLogger("VHDS")

    self.crypto = CryptoAnalyzer()
    self.entropy = EntropyAnalyzer()
    self.classifier = HashClassifier()

def analyze_hash(self, h: str) -> VHDSResult:
    """
    Повний аналіз одного хешу.
    """
    self.logger.info("Аналіз хешу: %s", h)

    crypto_profile = self.crypto.analyze(h)
    entropy_profile = self.entropy.analyze(h)

    features = self.classifier.features_from_profiles(
        length=crypto_profile.length,
        entropy=entropy_profile.entropy,
        unique_chars=entropy_profile.unique_chars,
        repeat_ratio=entropy_profile.repeat_ratio,
    )
    ml_risk = self.classifier.predict_risk(features)

    result = VHDSResult(
        raw=crypto_profile.raw,
        suspected_algorithm=crypto_profile.suspected_algorithm,
        crypto_risk=crypto_profile.risk_level,
        ml_risk=ml_risk,
        length=crypto_profile.length,
        entropy=entropy_profile.entropy,
        unique_chars=entropy_profile.unique_chars,
        repeat_ratio=entropy_profile.repeat_ratio,
        alphabet_type=crypto_profile.alphabet_type,
```

```
        )

        self.logger.info(
            "Результат: algo=%s crypto_risk=%s ml_risk=%s",
            result.suspected_algorithm,
            result.crypto_risk,
            result.ml_risk,
        )
        return result

    def analyze_many(self, hashes: List[str]) -> List[VHDSResult]:
        """
        Пакетний аналіз списку хешів.
        """
        return [self.analyze_hash(h) for h in hashes if h.strip()]

# 7. МОДУЛЬ ФОРМУВАННЯ ЗВІТІВ
# ---------------------------

class ReportManager:
    """
    Модуль формування звітів.
    Зберігає результати аналізу у форматах JSON та CSV для аудиту.
    """

    def __init__(self, reports_dir: str = "reports") -> None:
        self.reports_dir = Path(reports_dir)
        self.reports_dir.mkdir(exist_ok=True)

    def save_report(self, results: List[VHDSResult]) -> Path:
        """
        Збереження звіту у файли JSON і CSV з часовою міткою.
        """
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
```

```
        json_path = self.reports_dir /
    f"vhds_report_{timestamp}.json"
        csv_path = self.reports_dir / f"vhds_report_{timestamp}.csv"

        rows = [asdict(r) for r in results]

        # JSON-звіт
        with json_path.open("w", encoding="utf-8") as f:
            json.dump(rows, f, ensure_ascii=False, indent=2)

        # CSV-звіт
        if rows:
            with csv_path.open("w", encoding="utf-8", newline="") as
    f:
                writer = csv.DictWriter(f,
    fieldnames=rows[0].keys())
                writer.writeheader()
                writer.writerows(rows)

        return json_path

    def get_last_report(self) -> Optional[str]:
        """
        Повертає вміст останнього JSON-звіту як рядок.
        """
        reports =
    sorted(self.reports_dir.glob("vhds_report_*.json"))
        if not reports:
            return None
        return reports[-1].read_text(encoding="utf-8")

# 8. REST API (FASTAPI) ДЛЯ ІНТЕГРАЦІЇ З КОРПОРАТИВНИМИ СИСТЕМАМИ
# ------------------------------------------------------------

app = FastAPI(title="VHDS API")
```

```
pipeline = VHDSPipeline()
reporter = ReportManager()

class AnalyzeRequest(BaseModel):
    """
    Вхідна структура для REST-методу /analyze.
    """
    hashes: List[str]


class AnalyzeResponseItem(BaseModel):
    """
    Вихідна структура для REST-методу /analyze.
    """
    raw: str
    suspected_algorithm: Optional[str]
    crypto_risk: str
    ml_risk: str
    length: int
    entropy: float
    unique_chars: int
    repeat_ratio: float
    alphabet_type: str

@app.get("/health")
def health() -> Dict[str, str]:
    """
    Проста перевірка доступності сервісу.
    """
    return {"status": "ok"}

@app.post("/analyze", response_model=List[AnalyzeResponseItem])
def analyze_hashes(req: AnalyzeRequest) ->
    List[AnalyzeResponseItem]:
```

```
    """
    REST-ендпоїнт для аналізу списку хешів.
    Повертає результати у структурованому вигляді та формує звіт.
    """
    results: List[VHDSResult] = pipeline.analyze_many(req.hashes)
    reporter.save_report(results)
    return [AnalyzeResponseItem(**r.__dict__) for r in results]


@app.get("/reports/last")
def get_last_report() -> Dict[str, str]:
    """
    REST-ендпоїнт для отримання останнього JSON-звіту.
    """
    data = reporter.get_last_report()
    if data is None:
        return {"message": "Звітів ще немає"}
    return {"report_json": data}
```