МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему

СИСТЕМА ОЦІНКИ ЕФЕКТИВНОСТІ МЕТОДІВ ШИФРУВАННЯ ДЛЯ ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ ДАНИХ

Виконав: студент групи 1КІ-23
Спеціальності 123 «Комп'ютерна інженерія»
Каращук О. М.
Керівник роботи
к.т.н., доцент Захарова М.В.
Кількість балів: _____
Оцінка: ECTS _____

Черкаси, 2025

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Кафедра комп'ютерної інженерії та інформаційних технологій

(повна назва випускової кафедри)

Спеціальність 123 «Комп'ютерна інженерія»

(шифр і назва спеціальності)

Освітня програма Комп'ютерна інженерія

(назва освітньої програми)

ЗАТВЕРДЖУЮ

Завідувач кафедри

комп'ютерної інженерії та

інформаційних технологій

(назва кафедри)

_____ Хотунов В.І.

(підпис) (ПІБ)

«_____»_____2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Каращуку Олександру Миколайовичу

(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи Система оцінки ефективності методів шифрування для забезпечення конфіденційності даних

Науковий керівник роботи Захарова Марія В'ячеславівна, к.т.н доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від "___" _____ 202_ року № ___.

2. Строк подання студентом випускної роботи 20.05.2025

3. Вихідні дані до випускної роботи Аналіз методів шифрування та критерії їх оцінки, визначення технологій для створення системи оцінки ефективності методів шифрування, програмна реалізація системи оцінки шифрування.

4. Зміст випускної роботи (перелік питань, які потрібно розробити) визначення актуальності теми, мети, завдання, об'єкту, предмету, аналіз методів шифрування, реалізація системи оцінки методів шифрування.

5. Дата видачі завдання 20.09.2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання з підписами наукового керівника і студента
1	Вступ	20.10.2024	
2	Розділ 1. Аналіз методів шифрування та визначення критеріїв їх оцінки	10.12.2024	
3	Розділ 2. Визначення технологій для створення системи оцінки ефективності методів шифрування	13.02.2025	
4	Розділ 3. Реалізація системи оцінки ефективності методів шифрування	30.04.2025	
5	Висновки	11.05.2025	
6	Оформлення випускної роботи (чистовий варіант)	17.05.2025	
7	Здача випускної роботи на кафедру для рецензування (за 14 днів до захисту)	05.06.2025	
8	Перевірка випускної роботи на наявність ознак плагіату (за 10 днів до захисту)	09.06.2025	
9	Подання випускної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	12.06.2025	

Студент _____ Каращук О.М.

(підпис) (прізвище та ініціали)

Науковий керівник роботи _____ Захарова М.В.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

З розвитком цифрових технологій та зростанням кількості кіберзагроз питання захисту конфіденційної інформації набуває особливої актуальності. У даній роботі розглянуто проблему вибору ефективного методу шифрування для забезпечення конфіденційності даних в інформаційних системах. Проаналізовано основні типи алгоритмів шифрування – симетричні, асиметричні, гібридні, та визначено основні критерії оцінки ефективності алгоритмів шифрування.

У роботі запропоновано підхід до автоматизованої оцінки ефективності криптоалгоритмів на основі методів машинного навчання. Розроблено клієнт-серверну систему, яка дозволяє користувачеві вводити параметри, що характеризують умови використання, та автоматично отримувати рекомендований алгоритм шифрування за допомогою багатокласової моделі класифікації. В якості інструментів реалізації використано технології ASP.NET та ML.NET.

Окрему увагу приділено архітектурі системи, принципам її роботи, побудові навчального датасету та оцінці результатів моделі. Проведено тестування системи на практичних сценаріях для перевірки її ефективності, точності класифікації та гнучкості застосування.

Отримані результати засвідчують доцільність використання машинного навчання для вирішення задач криптографічного захисту інформації та дозволяють зменшити час на вибір оптимального алгоритму шифрування з урахуванням специфіки конкретного випадку застосування.

ABSTRACT

With the development of digital technologies and the increase in the number of cyber threats, the issue of protecting confidential information is becoming particularly relevant. This paper considers the problem of choosing an effective encryption method to ensure data confidentiality in information systems. The main types of encryption algorithms are analyzed – symmetric, asymmetric, hybrid, and the main criteria for evaluating the effectiveness of encryption algorithms are determined.

The paper proposes an approach to automated evaluation of the effectiveness of crypto algorithms based on machine learning methods. A client-server system has been developed that allows the user to enter parameters that characterize the conditions of use and automatically receive the recommended encryption algorithm using a multi-class classification model. ASP.NET and ML.NET technologies have been used as implementation tools.

Special attention is paid to the architecture of the system, the principles of its operation, the construction of the training dataset and the evaluation of the model results. The system has been tested on practical scenarios to verify its effectiveness, classification accuracy and application flexibility.

The results obtained demonstrate the feasibility of using machine learning to solve cryptographic information protection problems and allow reducing the time required to select the optimal encryption algorithm, taking into account the specifics of a particular application.

ЗМІСТ

ВСТУП 4

РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ ШИФРУВАННЯ ТА ВИЗНАЧЕННЯ КРИТЕРІЇВ ЇХ ОЦІНКИ 7

1.1 Роль шифрування у захисті даних 7

1.2 Симетричні методи шифрування 8

1.2.1 Метод шифрування AES 9

1.2.2 Метод шифрування ChaCha20 11

1.3 Асиметричні методи шифрування 14

1.3.1 Метод шифрування RSA 15

1.3.2 Метод шифрування ECC 18

1.4 Гібридні методи шифрування 20

1.5 Основні критерії оцінки ефективності алгоритмів шифрування 21

1.6 Актуальні загрози безпеці та проблеми вибору оптимального методу шифрування 32

РОЗДІЛ 2 ВИЗНАЧЕННЯ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СИСТЕМИ ОЦІНКИ ЕФЕКТИВНОСТІ МЕТОДІВ ШИФРУВАННЯ 37

2.1 Аналіз вимог до системи 37

2.2 Обґрунтування вибору архітектури клієнт-сервер 37

2.3 Стек технологій для реалізації системи 39

2.3.1 Технології клієнтської частини 39

2.3.2 Технології серверної частини 40

2.4 Використання машинного навчання для автоматичного вибору методу шифрування 42

2.4.1 Використання ML.NET для автоматизації вибору шифрування 43

2.4.2 Аналіз методів машинного навчання для задачі вибору шифрування 44

РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ ОЦІНКИ ЕФЕКТИВНОСТІ МЕТОДІВ ШИФРУВАННЯ 48

3.1 Загальна архітектура системи 48

3.2 Розробка користувацького інтерфейсу 49

3.3 Логіка роботи моделі машинного навчання 50

3.4 Розробка серверної частини 54

3.5 Тестування роботи системи 64

ВИСНОВКИ 71

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 73

ВСТУП

Сучасні інформаційні технології відіграють ключову роль у всіх сферах суспільного життя, забезпечуючи швидкий обмін даними та автоматизацію процесів. Проте зростання обсягів цифрової інформації та її передача через відкриті канали зв'язку створюють серйозні виклики для забезпечення безпеки даних. Конфіденційність інформації є однією з основних вимог до сучасних інформаційних систем, адже порушення захисту може призвести до значних фінансових, репутаційних та безпекових втрат. У цьому контексті методи шифрування виступають основним інструментом захисту даних, а вибір оптимального алгоритму шифрування є складною задачею, що вимагає врахування багатьох факторів, таких як продуктивність, безпека та ресурсоємність.

Актуальність. У сучасних умовах стрімкого зростання обсягів цифрових даних забезпечення конфіденційності інформації є критично важливим завданням кібербезпеки. Збільшення кількості кібератак, зокрема витоків даних, перехоплення інформації та атак на криптографічні системи, підкреслює необхідність ефективного вибору методів шифрування. Традиційні підходи до вибору алгоритмів шифрування часто є суб'єктивними, не враховують динамічних умов експлуатації та не забезпечують оптимального співвідношення безпеки, продуктивності й ресурсоємності. Розробка системи автоматизованого вибору методів шифрування на основі машинного навчання є актуальним рішенням, адже нині відсутні універсальні та зручні засоби для швидкого, автоматизованого вибору найдоцільнішого методу шифрування, орієнтованого на задані параметри, а наявні підходи часто базуються на загальних рекомендаціях без урахування специфіки конкретного випадку. Вирішення цієї задачі сприятиме підвищенню рівня захищеності даних та ефективності роботи інформаційних систем.

Мета роботи. Розробка системи оцінки ефективності методів шифрування для забезпечення конфіденційності даних шляхом автоматизації вибору оптимального алгоритму з використанням моделі машинного навчання.

Завдання роботи:

- Проаналізувати сучасні методи шифрування (симетричні, асиметричні, гібридні) та визначити критерії їх оцінки.
- Обґрунтувати вибір архітектури та технологій для створення системи оцінки ефективності.
- Розробити клієнт-серверну систему з інтеграцією моделі машинного навчання для автоматичного вибору алгоритму шифрування.
- Реалізувати користувацький інтерфейс і серверну частину системи.
- Провести тестування системи для оцінки її функціональності та ефективності.

Об’єкт дослідження. Процес вибору криптографічного методу у системах захисту інформації.

Предметом дослідження є методи оцінки ефективності алгоритмів шифрування та їх автоматизований вибір на основі технологій машинного навчання.

Методи дослідження. Для реалізації поставлених завдань використовувались методи теоретичного аналізу та узагальнення наукових джерел, системного аналізу вимог, порівняльного аналізу криптоалгоритмів, методи машинного навчання для класифікації та прогнозування, а також інструменти практичного програмування, зокрема ASP.NET, ML.NET та технології для розробки вебінтерфейсу.

Інформаційна база. У роботі використано наукові статті з питань криптографії, технічну документацію бібліотеки ML.NET, стандарти шифрування (AES, ChaCha20, RSA, ECC), аналітичні матеріали щодо сучасних загроз кібербезпеці, а також загальнодоступні ресурси з прикладами застосування ML.NET у подібних задачах.

Практичне значення. Розроблена система може застосовуватися в інформаційних системах для автоматизації вибору методів шифрування, що підвищує ефективність захисту даних. Вона забезпечує гнучкість у виборі алгоритмів, зменшує час на їх конфігурацію та може бути використана в веб-

додатках, банківських системах і інших галузях, де захист даних є критично важливим.

Структура і обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та списку використаних джерел (30 найменувань). Загальний обсяг роботи становить 79 сторінок комп’ютерного тексту, 18 рисунків, 17 формул.

РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ ШИФРУВАННЯ ТА ВИЗНАЧЕННЯ КРИТЕРІЇВ ЇХ ОЦІНКИ

1.1 Роль шифрування у захисті даних

Шифрування відіграє ключову роль у захисті даних, забезпечуючи їхню конфіденційність, цілісність та доступність лише для авторизованих користувачів. У сучасному цифровому світі інформація постійно передається через відкриті мережі, зберігається у хмарних сервісах, використовується у фінансових транзакціях, корпоративних системах та державних структурах. Без належного захисту ці дані можуть бути перехоплені, змінені або використані зловмисниками, що призводить до фінансових втрат, витоку конфіденційної інформації чи навіть загроз національній безпеці. Шифрування дозволяє захистити дані від несанкціонованого доступу, перетворюючи їх у зашифрований вигляд, який неможливо розшифрувати без відповідного ключа. Це забезпечує конфіденційність, оскільки навіть у разі перехоплення інформації зловмисник не зможе її прочитати без знання алгоритму та ключа дешифрування. Завдяки цьому шифрування використовується у всіх сферах, де є потреба в захисті переданої або збереженої інформації, зокрема в онлайн-банкінгу, електронній пошті, месенджерах, електронному документообігу та системах доступу до конфіденційних даних.

Окрім конфіденційності, шифрування забезпечує цілісність даних, унеможливлюючи їх зміну або підробку без виявлення. Наприклад, якщо зловмисник спробує змінити зашифровані дані, це призведе до того, що при дешифруванні інформація стане непридатною для використання. Багато сучасних систем використовують алгоритми шифрування в поєднанні з механізмами контролю цілісності, такими як хеш-функції та цифрові підписи, що дозволяє виявляти будь-які спроби модифікації інформації. Шифрування також відіграє важливу роль у процесах автентифікації та ідентифікації користувачів. Багато систем, зокрема онлайн-банкінг та корпоративні VPN, використовують криптографічні механізми для підтвердження особи користувача перед наданням

доступу до інформації. Це може здійснюватися за допомогою криптографічних ключів, сертифікатів безпеки або цифрових підписів, що гарантують, що дані отримує саме той, кому вони призначені [1].

У сучасних реаліях, коли кібератаки стають дедалі витонченішими, а обчислювальні потужності зловмисників зростають, шифрування стає невід'ємною частиною інформаційної безпеки. Використання надійних криптографічних алгоритмів дозволяє захищати персональні дані користувачів, забезпечувати безпечні транзакції та запобігати витоку важливої інформації. Однак ефективність шифрування залежить від правильного вибору алгоритмів, довжини ключів, механізмів керування ключами та рівня обчислювальних ресурсів, необхідних для його реалізації.

1.2 Симетричні методи шифрування

Симетричні методи шифрування – це клас криптографічних алгоритмів, у яких для шифрування і дешифрування використовується один і той самий ключ. Це означає, що як відправник, так і отримувач даних повинні мати доступ до цього ключа для коректного обміну інформацією. Основний принцип роботи симетричних алгоритмів полягає у перетворенні відкритого тексту в зашифрований вигляд за допомогою ключа та відповідного математичного алгоритму, після чого цей самий ключ використовується для відновлення вихідного тексту. Симетричне шифрування може бути реалізоване у вигляді потокових або блочних алгоритмів. Потокові шифри працюють з даними послідовно, обробляючи їх біт за бітом або байт за байтом, що робить їх ефективними для захисту потокової інформації, такої як голосові або відеозаписи. Блочні шифри, навпаки, оперують фіксованими блоками даних, наприклад, 64 або 128 біт, що забезпечує вищий рівень захисту та передбачувану структуру зашифрованої інформації.

Головною перевагою симетричних алгоритмів є їхня висока швидкість роботи. Завдяки меншій обчислювальній складності порівняно з асиметричними методами, вони дозволяють ефективно обробляти великі обсяги інформації, що робить їх ідеальним вибором для шифрування файлових сховищ, баз даних та

інших ресурсомістких систем. Симетричні алгоритми також менш вимогливі до апаратного забезпечення, що робить їх придатними для використання навіть на пристроях із обмеженими обчислювальними можливостями, таких як мобільні пристрої або вбудовані системи.

Проте основним недоліком симетричного шифрування є проблема безпечного обміну ключами. Оскільки однаковий ключ використовується як для шифрування, так і для дешифрування, необхідно знайти надійний спосіб його передачі між користувачами. Якщо зловмисник перехопить ключ, він зможе безперешкодно розшифрувати всю зашифровану інформацію [3]. Для розв'язання цієї проблеми в багатьох системах використовують додаткові механізми безпеки, такі як захищені канали передачі або гібридні криптографічні системи, де симетричний ключ передається за допомогою асиметричного шифрування.

Симетричні алгоритми широко застосовуються в різних сферах, де потрібне швидке та ефективне шифрування даних. Вони використовуються у протоколах безпеки, таких як SSL/TLS для захисту інтернет-з'єднань, у VPN-сервісах для шифрування трафіку, в банківських системах для захисту фінансових транзакцій, у системах збереження даних та в програмному забезпеченні для захисту конфіденційних файлів.

1.2.1 Метод шифрування AES

AES (Advanced Encryption Standard) – це один із найнадійніших і найпоширеніших методів симетричного шифрування, який використовується для захисту даних. Його було прийнято як стандарт шифрування в 2001 році Національним інститутом стандартів і технологій США. Алгоритм розроблений для забезпечення високого рівня безпеки з урахуванням сучасних вимог до швидкодії та стійкості до зломів.

Метод шифрування AES працює з блоками даних розміром 128 біт. Вибір ключа для шифрування може бути 128, 192 або 256 біт, що визначає кількість раундів обробки: 10 раундів для ключа 128 біт, 12 – для 192 біт, і 14 – для 256 біт. AES забезпечує високу стійкість до атак завдяки своїй складній структурі і

численним етапам обробки даних. Шифрування в AES проходить через декілька етапів, серед яких ключові операції включають SubBytes, ShiftRows, MixColumns, і AddRoundKey. Під час першого етапу, SubBytes, кожен байт блоку даних замінюється відповідно до значення в фіксованій таблиці S-Box. Операція підстановки може бути виражена через формулу (1.1):

$$S(b) = S\text{-}Box[b] \tag{1.1}$$

де $S(b)$ – це байт після підстановки;

S-Box – таблиця підстановки, яка містить статичні значення для кожного байта;

b – байт вхідних даних.

Далі застосовується операція ShiftRows, де рядки блоків зміщуються на певну кількість позицій. Наприклад, перший рядок не змінюється, другий зміщується на одну позицію, третій – на дві і так далі. Це виражається через формулу (1.2):

$$ShiftRows(R) = [R_0, R_1, R_2, R_3] \tag{1.2}$$

де R_n – це рядки матриці, які зміщуються.

Наступною операцією є MixColumns, де стовпці блоку перетворюються за допомогою матричного множення. Цей етап відповідає за змішування даних між стовпцями. Формула для цього виглядає так (1.3):

$$MixColumns(C) = C \times M \tag{1.3}$$

де C – це стовпець даних;

M – матриця, що використовується для множення.

Останнім етапом є AddRoundKey, при якому кожен блок даних зашифровується шляхом додавання (за допомогою операції XOR) до поточного стану блоку відповідного раундового ключа. Формула для цієї операції виглядає так (1.4):

$$S' = S \oplus K \tag{1.4}$$

де S – поточний стан блоку;

K – раундовий ключ;

$\oplus$ – операція XOR.

Ці операції виконуються на кожному раунді до тих пір, поки всі раунди не завершаться, і результатом буде зашифроване повідомлення [4]. Процес зворотного шифрування складається з тих самих операцій, але в зворотному порядку.

Однією з головних переваг AES є його висока безпека. Завдяки складній математичній структурі та багатоступеневій обробці, AES є стійким до відомих атак, таких як брутфорс чи аналіз даних. Алгоритм також відзначається універсальністю: його можна використовувати як для шифрування невеликих обсягів даних (наприклад, повідомлень), так і для масивних інформаційних потоків у реальному часі. Ще однією перевагою є швидкодія, особливо на сучасному обладнанні, яке підтримує апаратне прискорення AES, наприклад, в процесорах Intel.

Проте AES має і свої недоліки. Один із них – залежність від безпечного управління ключами. Якщо ключ буде скомпрометовано, вся зашифрована інформація стане вразливою. Інший недолік стосується роботи з великими обсягами даних у потоковому режимі. Хоча AES підтримує режими роботи для потокового шифрування, налаштування таких режимів може бути складним для некваліфікованих користувачів.

AES використовується в багатьох сферах, де потрібен високий рівень захисту даних. Він є основою для шифрування в протоколах HTTPS, VPN, TLS, які забезпечують безпеку інтернет-комунікацій. Також AES застосовується у файлових системах, наприклад BitLocker або VeraCrypt, для захисту даних на жорстких дисках і флеш-накопичувачах. Крім того, його широко використовують у мобільних додатках, банківських системах і хмарних сервісах для захисту конфіденційної інформації.

1.2.2 Метод шифрування ChaCha20

Метод шифрування ChaCha20 є поточним шифром, що базується на принципі генерації псевдовипадкових потоків для шифрування даних. Алгоритм ChaCha20 працює на основі 20 раундів перетворень, виконуючи серію операцій додавання, побітових зсувів і XOR, які забезпечують високу дифузію та нелінійність.

Початковим етапом роботи ChaCha20 є створення матриці стану розміром 4 x 4, яка включає чотири основні компоненти: константи, ключ, число блоку та нонс. Матриця стану S для ChaCha20 на початку шифрування має вигляд (1.5):

$$S = \begin{bmatrix} C_0 & C_1 & C_2 & C_3 \\ K_0 & K_1 & K_2 & K_3 \\ K_4 & K_5 & K_6 & K_7 \\ B & N_0 & N_1 & N_2 \end{bmatrix} \quad (1.5)$$

де C_i – константи;

K_i – 256-бітний ключ;

B – номер блоку;

N_i – 96-бітний нонс.

Основний цикл шифрування складається з 20 раундів, де кожен раунд складається з чергування «четвіркового» і «стовпчикового» перетворень. У кожному перетворенні використовується функція ChaCha20 Quarter Round, яка оновлює значення чотирьох елементів матриці стану. Функція Quarter Round визначається формулами (1.6):

$$a = a + b, d = (d \oplus \text{a}) \lll 16, \quad (1.6)$$
$$c = c + d, b = (b \oplus \text{c}) \lll 12,$$
$$a = a + b, d = (d \oplus \text{a}) \lll 8,$$
$$c = c + d, b = (b \oplus \text{c}) \lll 7$$

де a, b, c, d – елементи матриці стану, які беруть участь у раунді;

$\oplus$ – операція XOR;

$\lll$ – циклічний зсув вліво на вказану кількість бітів.

Після виконання 20 раундів оновлені значення додаються до початкового стану матриці, щоб згенерувати вихідний ключовий потік (1.7):

$$S_{out} = S_{in} + S_{rounds} \quad (1.7)$$

де S_{in} – початкова матриця стану;

S_{rounds} – матриця після 20 раундів.

Ключовий потік використовується для шифрування або дешифрування даних за допомогою операції XOR (1.8):

$$C_i = P_i \oplus \mathrm{K}_i \quad (1.8)$$

де $\mathrm{C_i}$ – шифротекст;

$\mathrm{P_i}$ – відкритий текст;

$\oplus$ – операція XOR;

$\mathrm{K_i}$ – ключовий потік.

Однією з основних переваг ChaCha20 є його висока безпека, яка забезпечується завдяки надійним криптографічним операціям. Відсутність серйозних вразливостей, що були властиві старішим алгоритмам шифрування, таких як RC4, робить ChaCha20 надійним вибором для використання в сучасних криптографічних протоколах. Крім того, алгоритм показує високу швидкість обробки даних, особливо на пристроях з обмеженими обчислювальними можливостями, таких як смартфони та планшети. Для порівняння, при використанні більш важких алгоритмів, таких як AES, потужність обчислювальних пристроїв може бути виснажена, особливо на мобільних пристроях. ChaCha20 ж дозволяє забезпечити високу продуктивність при відносно низьких вимогах до ресурсів [5].

Незважаючи на численні переваги, ChaCha20 також має кілька недоліків. Оскільки він є потоковим шифром, його слід використовувати лише в тих випадках, де потік даних можна обробляти по черзі, що не завжди підходить для всіх типів застосунків. Наприклад, для великих обсягів даних у вигляді файлів, коли важлива не тільки безпека, а й швидкість обробки, блочні шифри можуть бути більш оптимальними. Крім того, ChaCha20 потребує велику кількість операцій над даними, що вимагає певних обчислювальних ресурсів для генерації потоку шифрування, хоча це не настільки критично для більшості сучасних пристроїв.

ChaCha20 знаходить застосування в багатьох сучасних криптографічних протоколах. Одним з найбільш значущих застосувань є вбудовування цього шифру в протокол TLS 1.3 для забезпечення захищеного з’єднання в Інтернеті. Він також використовується в системах VPN для захисту приватності при передачі даних через зашифровані канали, у криптовалютних гаманцях для забезпечення безпечної генерації ключів та підписів, а також у мобільних платформах, де ресурси

обмежені, а швидкість шифрування має критичне значення. Google використав ChaCha20 у своєму протоколі для безпеки передачі даних у мобільних додатках, замінивши RC4, який мав численні проблеми безпеки.

1.3 Асиметричні методи шифрування

Асиметричні методи шифрування, також відомі як криптографія з публічними ключами, є одним із основних підходів до забезпечення конфіденційності та цілісності даних у сучасних системах безпеки. На відміну від симетричних методів, які використовують один ключ для шифрування та дешифрування даних, асиметричні методи використовують пару ключів: публічний і приватний. Публічний ключ доступний усім, хто бажає надіслати зашифровану інформацію, тоді як приватний ключ залишається в таємниці й використовується для розшифрування отриманих даних. Ця особливість дозволяє вирішити проблему безпечного обміну ключами, яка є слабким місцем симетричного шифрування.

Принцип роботи асиметричних алгоритмів полягає в тому, що публічний ключ використовується для шифрування даних, а відповідний приватний ключ – для їхнього розшифрування. Така схема забезпечує високий рівень безпеки, оскільки навіть якщо зловмисник перехопить публічний ключ, він не зможе розшифрувати інформацію без доступу до приватного ключа [7]. Проте, процес шифрування та дешифрування в асиметричних методах є значно повільнішим, порівняно з симетричними методами, через складність математичних операцій, що використовуються для шифрування.

Основною перевагою асиметричних методів є можливість забезпечити захист даних навіть без необхідності попереднього обміну секретним ключем, що робить їх ідеальними для сценаріїв, де важлива безпека і зручність, наприклад, для електронних підписів або обміну інформацією через відкриті канали зв’язку. Ще однією перевагою є те, що публічний ключ можна поширювати широко, а приватний ключ залишається захищеним, що дозволяє користувачам забезпечити безпеку своїх даних навіть в умовах відкритого середовища, як Інтернет. Крім того,

асиметричні алгоритми забезпечують цілісність і автентичність даних, адже електронний підпис, що базується на асиметричному шифруванні, підтверджує, що дані не були змінені під час передачі.

Однак асиметричне шифрування має і свої недоліки. Найбільшим є його повільність порівняно з симетричним шифруванням. Оскільки для обробки кожного повідомлення потрібні складні математичні операції, це може значно збільшити час обробки, особливо при роботі з великими обсягами даних. Через це асиметричне шифрування часто застосовується не для шифрування великих обсягів даних, а переважно для захисту ключів у гібридних системах шифрування, де асиметричні методи використовуються для безпечного обміну симетричними ключами. Крім того, асиметричні алгоритми вимагають більше обчислювальних ресурсів, що може бути обмеженням для деяких пристроїв або систем з низькою продуктивністю.

Асиметричні методи шифрування широко застосовуються в таких сферах, як електронний підпис, забезпечення безпеки електронної пошти, створення безпечних з'єднань через SSL/TLS для захисту веб-трафіку, а також для забезпечення конфіденційності в системах онлайн-банкінгу та електронних грошових переказів. Водночас, протоколи, що використовують асиметричні методи, зазвичай поєднують їх з симетричним шифруванням для досягнення кращої швидкості обробки даних і високої рівні безпеки.

1.3.1 Метод шифрування RSA

RSA (Rivest-Shamir-Adleman) – це один із найбільш відомих і широко використовуваних алгоритмів асиметричного шифрування. Основна ідея RSA полягає в тому, що кожен користувач має пару ключів: публічний і приватний. Публічний ключ використовується для шифрування інформації, а приватний – для її розшифрування. Ключі пов'язані між собою математичним чином, що робить можливим процес шифрування і розшифрування лише за допомогою відповідних ключів, при цьому навіть якщо публічний ключ доступний всім, він не дозволяє безпосередньо отримати приватний.

Принцип роботи RSA базується на складності факторизації великих чисел. Алгоритм RSA працює в кілька етапів: генерація ключів, шифрування даних і розшифрування. На етапі генерації ключів вибирають два великі прості числа p і q. На основі добутку цих чисел обчислюється модуль n, який використовується для шифрування і розшифрування. Після цього обчислюється значення функції Ейлера (1.9):

$$\phi(n) = (p - 1) \times (q - 1) \quad (1.9)$$

де $\phi(n)$ визначає кількість чисел менших за n, які є взаємно простими з n;

p, q – два великі прості числа.

Далі вибирають відкрите експоненційне число e, яке задовольняє умови (1.10):

$$1 < e < \phi(n), \text{НОД}(e, \phi(n)) = 1 \quad (1.10)$$

де e – відкрите експоненційне число;

$\phi(n)$ – значення функції Ейлера;

НОД – найбільний спільний дільник.

Використовуючи e, знаходять закрите експоненційне число d, яке є мультиплікативним оберненим до e за модулем $\phi(n)$ (1.11):

$$e \times d \equiv 1 (mod\ \phi(n)) \quad (1.11)$$

де e – відкритий експонент;

d – закритий експонент;

$\phi(n)$ – значення функції Ейлера.

$\phi(n)$ – функція Ейлера, що визначає модуль для обчислення оберненого числа.

Процес шифрування здійснюється шляхом перетворення повідомлення M у числове представлення, яке менше за n. Шифротекст C обчислюється за формулою (1.12):

$$C = M^e mod\ n \quad (1.12)$$

де C – шифротекст;

M – відкритий текст;

e і n – параметри відкритого ключа.

Розшифрування виконується аналогічно за допомогою закритого ключа (1.13):

$$M = C^d mod\ n \quad (1.13)$$

де M – відкритий текст;

C – шифротекст;

d – закрите експоненційне число;

n – параметр відкритого ключа.

Безпека RSA базується на складності факторизації великого числа n, яке є добутком двох простих чисел p і q. Якщо зловмисник не може розкласти n на множники, він не зможе обчислити ϕ(n) і, відповідно, знайти значення d.

Переваги RSA включають високий рівень безпеки при великих розмірах ключів, а також широке застосування в різних системах і протоколах. Завдяки тому, що цей алгоритм є асиметричним, RSA дозволяє забезпечити конфіденційність даних без необхідності попереднього обміну секретними ключами. Він також забезпечує можливість використання цифрових підписів, що дозволяє автентифікувати відправника і перевірити цілісність повідомлення. RSA став основою для багатьох сучасних систем безпеки, включаючи протоколи SSL/TLS, які використовуються для захисту з'єднань в Інтернеті.

Однак RSA має й низку недоліків. Одним з основних є те, що при великих розмірах ключів обчислення можуть бути дуже ресурсоємними, що знижує ефективність алгоритму, особливо при обробці великих обсягів даних. Наприклад, для досягнення високого рівня безпеки RSA потребує використання ключів довжиною 2048 біт і більше, що може призвести до значних затримок у процесі шифрування і розшифрування [9]. Крім того, RSA вимагає значних обчислювальних ресурсів, що робить його менш ефективним для мобільних пристроїв або інших систем з обмеженими можливостями.

RSA використовується в численних сферах, де необхідно забезпечити захист інформації. Наприклад, він є основою для протоколів безпеки в Інтернеті, таких як HTTPS, який забезпечує безпечний обмін даними між веб-сайтами і користувачами. Також RSA широко використовується для створення цифрових

підписів, які застосовуються для підтвердження автентичності електронних документів і програмного забезпечення. Крім того, цей метод шифрування застосовується в електронних платіжних системах, криптовалютах, а також у багатьох корпоративних системах для забезпечення безпеки даних при передачі через незахищені канали зв'язку.

1.3.2 Метод шифрування ECC

ECC (Elliptic Curve Cryptography) – це метод шифрування, заснований на використанні еліптичних кривих для забезпечення безпеки при обміні даними. Він є сучасним і дуже ефективним методом асиметричного шифрування, який дозволяє використовувати коротші ключі для досягнення рівня безпеки, подібного до інших методів, таких як RSA, але при цьому потребує значно менше ресурсів. Основою ECC є використання еліптичних кривих, визначених рівнянням (1.14):

$$y^2 = x^3 + ax + b(mod\ p) \quad (1.14)$$

де y, x – координати точок на кривій;

a, b – коефіцієнти, які визначають форму кривої;

p – просте число, що задає поле.

Важливо, що параметри a і b мають бути обрані так, щоб забезпечити відсутність особливих точок на кривій.

Для ECC використовуються математичні операції над точками кривої, серед яких ключовою є операція множення точки P (базова точка, фіксована для всіх учасників системи), на скаляр k (ціле число, що відповідає приватному ключу), результатом якої є точка Q. Для генерації ключової пари ECC обирається приватний ключ d, а відповідний публічний ключ Q обчислюється як добуток d на P. Процес шифрування повідомлення в ECC відбувається наступним чином. Спочатку повідомлення M перетворюється на точку на еліптичній кривій. Відправник вибирає випадкове число k і обчислює першу точку, виконуючи добуток випадкового числа на базову точку кривої (1.15):

$$C_1 = kP \quad (1.15)$$

де k – випадкове число;

P – базова точка кривої.

Далі відправник бере повідомлення у вигляді точки на еліптичній кривій і обчислює другу точку шифротексту. Для цього він додає до цієї точки результат множення публічного ключа одержувача на вибране випадкове число (1.16):

$$C_2 = M + kQ \tag{1.16}$$

де M – точка на еліптичній кривій;

k – випадкове число;

Q – публічний ключ одержувача.

Шифротекст складається з двох точок: одна є результатом множення випадкового числа на базову точку, а друга – результатом додавання до повідомлення точки, що отримана множенням публічного ключа одержувача на випадкове число. Одержувач, знаючи свій приватний ключ d, може розшифрувати повідомлення, виконавши операцію (1.17):

$$M = C_2 - dC_1 \tag{1.17}$$

де C_2 – друга точка шифротексту;

d – приватний ключ одержувача;

C_1 – перша точка шифротексту.

Однією з основних переваг ECC є її ефективність: для досягнення рівня безпеки, подібного до RSA, використовуються значно менші ключі. Наприклад, для забезпечення рівня безпеки в 128 біт (що є стандартом для багатьох сучасних криптографічних систем) у ECC достатньо використовувати ключ довжиною лише 256 біт, тоді як для RSA потрібно близько 3072 біт. Це означає, що ECC є набагато менш ресурсомістким, що робить його ідеальним для застосування в мобільних пристроях, IoT (Інтернет речей) та інших системах з обмеженими ресурсами [10]. Завдяки меншій потребі в обчислювальних ресурсах ECC також може забезпечити вищу швидкість роботи порівняно з іншими методами.

Незважаючи на численні переваги, ECC має і деякі недоліки. Одним з них є складність реалізації та вивчення математики еліптичних кривих, що може бути проблемою для деяких розробників, які не мають відповідної підготовки. Окрім того, з-за новизни технології вона ще не настільки широко підтримується у

порівнянні з RSA, тому впровадження в деяких системах може вимагати додаткових зусиль. Також важливо відзначити, що, незважаючи на свою ефективність, ECC може бути вразливою до атак, якщо вибір кривої або реалізація алгоритму не є достатньо надійними.

ECC застосовується в різних сферах, зокрема в мобільних пристроях, де ефективність і швидкість мають велике значення. Багато сучасних протоколів безпеки, таких як TLS та VPN, використовують ECC для забезпечення швидкого і надійного шифрування. Також ECC активно використовується в криптовалютах, таких як Bitcoin, для забезпечення безпеки транзакцій і створення цифрових підписів.

1.4 Гібридні методи шифрування

Гібридні методи шифрування – це підхід, який поєднує переваги як симетричних, так і асиметричних методів шифрування. Вони були розроблені для того, щоб використовувати сильні сторони кожного з цих методів і подолати їх недоліки. Принцип роботи гібридних методів полягає в тому, що для обміну ключами використовується асиметричне шифрування, а для самого шифрування даних – симетричне. Така комбінація дозволяє ефективно забезпечувати як безпеку, так і високу швидкість обробки даних.

У гібридних системах спочатку генерується симетричний ключ, який використовується для шифрування великого обсягу даних. Після цього цей ключ шифрується за допомогою асиметричного методу, тобто публічний ключ отримувача застосовується для захисту симетричного ключа. Оскільки асиметричне шифрування є повільнішим, його використовують лише для захисту маленького обсягу даних – самих ключів шифрування, що дозволяє досягти оптимальної швидкості передачі інформації. Коли отримувач отримує зашифрований симетричний ключ, він розшифровує його за допомогою свого приватного ключа, після чого може дешифрувати повідомлення за допомогою симетричного алгоритму [11].

Основною перевагою гібридних методів є те, що вони поєднують високу ефективність симетричного шифрування для шифрування великих обсягів даних з високим рівнем безпеки, який забезпечується асиметричними алгоритмами для обміну ключами. Цей підхід дозволяє зберегти баланс між швидкістю та безпекою, що особливо важливо при роботі з великими обсягами даних в реальному часі. Крім того, гібридні методи надають можливість безпечного обміну ключами без необхідності попередньої домовленості між сторонами, що забезпечує додаткову зручність.

Однак, незважаючи на всі переваги, гібридні методи мають і свої недоліки. Одним з основних є складність реалізації такої системи, оскільки необхідно поєднувати два різних алгоритми шифрування, що потребує додаткових обчислювальних ресурсів. Крім того, хоча гібридне шифрування забезпечує високий рівень безпеки, воно все ж є вразливим до атак, якщо один з компонентів (наприклад, асиметричний алгоритм) виявиться скомпрометованим. Також, через необхідність обробляти як симетричне, так і асиметричне шифрування, таке шифрування може бути більш повільним у порівнянні з використанням лише одного методу.

Гібридні методи шифрування широко використовуються в сучасних технологіях забезпечення безпеки, таких як SSL/TLS для захисту веб-трафіку в Інтернеті, де використовується асиметричне шифрування для обміну ключами і симетричне для шифрування даних. Крім того, вони використовуються в системах електронного банкінгу, електронних підписах, а також у багатьох протоколах для захисту персональних даних. Їх застосування є ключовим для забезпечення безпеки в умовах, коли необхідно обробляти великі обсяги даних, при цьому зберігаючи високу швидкість і рівень захисту.

1.5 Основні критерії оцінки ефективності алгоритмів шифрування

Оцінка ефективності алгоритмів шифрування є ключовим етапом у виборі найвідповіднішого методу для забезпечення конфіденційності даних у конкретних умовах. Вибір алгоритму не може ґрунтуватися лише на його популярності чи

частоті використання – необхідно враховувати низку технічних, практичних та безпекових характеристик, які безпосередньо впливають на загальну надійність та продуктивність криптографічного рішення. Серед основних критеріїв, які потрібно враховувати при оцінці алгоритмів шифрування, виділяються: рівень безпеки, швидкодія, платформа використання, обмеження по ресурсах, тип даних, що шифруються, тип шифрування, сумісність зі стандартами, масштабованість/підтримка паралельності, підтримка автентифікації та стійкість до помилок. Кожен з цих критеріїв має свої специфічні вимоги, що залежать від конкретних задач та умов використання системи. Кожен із цих аспектів впливає на вибір найбільш підходящого методу шифрування для конкретного випадку, тому детальний розгляд кожного з критеріїв дозволить краще зрозуміти, як оптимально вирішити задачу захисту даних.

«Рівень безпеки» є центральним критерієм при оцінці ефективності алгоритмів шифрування, оскільки саме він визначає здатність криптографічної системи забезпечувати захист конфіденційної інформації від несанкціонованого доступу. Цей критерій охоплює цілу низку технічних характеристик, які формують загальну криптостійкість алгоритму. Насамперед, це довжина криптографічного ключа, яка безпосередньо впливає на складність атаки перебором. Чим більша довжина ключа, тим більше комбінацій потрібно перевірити, що значно ускладнює задачу зловмисника. Однак ефективний рівень безпеки не обмежується лише кількістю бітів у ключі – важливим є також стійкість алгоритму до конкретних типів атак, таких як диференціальний і лінійний криптоаналіз, атаки з використанням обраного відкритого або зашифрованого тексту, атаки по сторонніх каналах, у тому числі часові та енергетичні атаки. До рівня безпеки також належить відсутність відомих криптографічних вразливостей у алгоритму. Якщо шифр був зламаний або продемонстровано ефективну атаку, яка знижує необхідну обчислювальну потужність для розшифрування, то навіть при великій довжині ключа його не можна вважати безпечним. Важливою характеристикою є й ступінь перевіреності алгоритму у науковій спільноті – алгоритми, що пройшли відкриту криптографічну експертизу і використовуються в міжнародних стандартах,

зазвичай мають вищий ступінь довіри. Рівень безпеки також враховує стійкість до еволюції обчислювальної техніки, зокрема до загроз, пов'язаних з розвитком квантових обчислень. Багато сучасних алгоритмів створювались без урахування квантових атак, тому вони можуть виявитися вразливими у майбутньому. У зв'язку з цим виникає потреба оцінювати алгоритм і з точки зору перспективи постквантової криптографії. Таким чином, якщо система працює в умовах з високими вимогами до конфіденційності, наприклад, державні органи, фінансові установи, зберігання медичних або військових даних, обирається високий рівень безпеки. У випадках, коли важливим є баланс між безпекою та продуктивністю, наприклад, у внутрішніх корпоративних системах або для захисту особистих даних пересічних користувачів, доцільним є вибір середнього рівня безпеки. Якщо ж ідеться про тимчасове або другорядне шифрування, де головним є швидкодія, а не криптостійкість, наприклад, шифрування кешованих даних або проміжна обробка на слабких пристроях, може бути допустимим і низький рівень безпеки, хоча такий підхід не рекомендується для конфіденційної інформації.

Одним із ключових критеріїв оцінки ефективності алгоритмів шифрування є «Швидкодія», особливо в контексті сучасних інформаційних систем, де обробка великих обсягів даних відбувається в режимі реального часу. Цей показник визначає, наскільки швидко алгоритм здатен виконувати операції шифрування та розшифрування інформації без значного навантаження на обчислювальні ресурси. Висока швидкодія є критично важливою для систем, що працюють в умовах обмежених затримок – наприклад, у стримінгових сервісах, фінансових транзакціях, онлайн-іграх або мережевих комунікаціях, де кожна додаткова мілісекунда може впливати на якість користувацького досвіду або загальну стабільність процесів. Під поняттям швидкодії мається на увазі не лише час, необхідний для обробки одного блоку даних, а й загальна ефективність виконання шифрувальних операцій на різних платформах – від високопродуктивних серверів до мобільних пристроїв з обмеженим енергоспоживанням і пам'яттю. Алгоритм має демонструвати стабільну продуктивність навіть при високому навантаженні або багатопоточній обробці. У процесі оцінки швидкодії враховуються також

затрати ресурсів на генерацію ключів, ініціалізацію криптографічного сеансу, обробку метаданих та інші супровідні операції, які можуть суттєво впливати на загальний час роботи системи. Деякі алгоритми демонструють високу швидкодію на програмному рівні, інші – ефективніші при апаратній реалізації, тому також враховується здатність алгоритму до апаратного прискорення, або спеціалізованих криптографічних модулів. Важливою є й енергетична ефективність при високій швидкості, оскільки надмірне споживання енергії внаслідок складності шифрувального процесу може стати критичним у пристроях з автономним живленням.

Такий критерій як «Платформа використання» відіграє важливу роль при виборі алгоритму шифрування, оскільки різні середовища мають власні технічні обмеження, архітектурні особливості та вимоги до ресурсів. Алгоритм, що демонструє високу ефективність у серверному середовищі, може виявитися надто важким або енергозатратним для мобільних пристроїв або вбудованих систем. Наприклад, мобільні платформи, такі як Android і iOS, мають обмежені обчислювальні ресурси та акумуляторну ємність, тому тут особливо важливі легковагові та енергоефективні алгоритми шифрування. До того ж мобільні додатки часто взаємодіють із сервером, що вимагає використання алгоритмів, які забезпечують оптимальне співвідношення між швидкістю та безпекою на обох кінцях з’єднання. У випадку веб-додатків і браузерів особливу увагу приділяють алгоритмам, сумісним із сучасними стандартами, такими як TLS, а також здатним ефективно працювати в умовах обмежень JavaScript або WebAssembly. Тут важливі як захист даних під час передавання, так і сумісність з великою кількістю клієнтських пристроїв, що вимагає використання алгоритмів, які підтримуються основними браузерами без додаткових компонентів. У серверних системах або хмарних платформах ключовими факторами є масштабованість, висока продуктивність під навантаженням і здатність до роботи в багатопоточному середовищі. У таких умовах перевага надається алгоритмам, які можуть бути реалізовані апаратно або оптимізовані під архітектуру CPU, що дозволяє знижувати затрати часу на шифрування великих масивів даних, забезпечуючи при цьому

високу безпеку. Хмари також часто працюють з даними різних користувачів, тому важливо підтримувати ефективні механізми управління ключами та ізоляції доступу. Для вбудованих систем і пристроїв Інтернету речей (IoT) основними є мінімальне енергоспоживання, компактність реалізації та можливість функціонування на мікроконтролерах з обмеженим обсягом пам’яті. Тут важливо використовувати шифрувальні алгоритми, які здатні працювати в умовах мінімального навантаження на систему, при цьому забезпечуючи достатній рівень захисту для передачі критично важливих даних, таких як показники сенсорів чи команди керування. Отже, критерій платформи використання охоплює здатність алгоритму адаптуватися до специфіки середовища, в якому він застосовується – від високопродуктивних серверів до енергообмежених мобільних або вбудованих пристроїв. Правильний вибір з урахуванням цього критерію забезпечує як ефективність, так і надійність роботи криптографічної системи в реальних умовах експлуатації.

Критерій «Обмеження по ресурсах» є надзвичайно важливим під час вибору алгоритму шифрування, особливо в умовах, коли система має обмежені технічні можливості або працює в енергозалежному середовищі. Насамперед, мова йде про мінімальне енергоспоживання, яке критично важливе для мобільних пристроїв, безпровідних сенсорів, пристроїв Інтернету речей (IoT) та інших вбудованих систем. У таких випадках алгоритм має виконуватись із мінімальним впливом на заряд акумулятора, що передбачає скорочення кількості обчислень, зменшення звернень до пам’яті та оптимізацію часу роботи процесора. Ще одним важливим аспектом є обмежена оперативна пам’ять. Пристрої з невеликим обсягом пам’яті не здатні обробляти складні або надто «важкі» алгоритми, що зберігають проміжні дані чи використовують великі блоки при обробці інформації. Тому ефективні алгоритми для таких середовищ мають бути компактними, споживати мінімум пам’яті та не потребувати складних структур для збереження ключів чи даних. Також важливо, щоб вони не створювали додаткове навантаження на систему в процесі тривалої роботи. Обмежена обчислювальна потужність є типовою характеристикою вбудованих процесорів або мікроконтролерів, які мають

невелике число ядер, низьку тактову частоту та відсутність апаратного прискорення криптографічних операцій. У такому випадку оптимальними є алгоритми, що розраховані на виконання з допомогою базових математичних операцій, без складної обробки великих чисел або багаторівневих перетворень. Це дозволяє зменшити час шифрування та розшифрування, що в свою чергу покращує продуктивність загальної системи. Водночас, існують середовища, в яких відсутні суттєві обмеження по ресурсах – наприклад, потужні сервери або хмарні платформи. У таких випадках можливо використовувати більш складні та захищені криптографічні алгоритми, які забезпечують максимальний рівень безпеки, навіть якщо це вимагає значних обчислювальних затрат. Це дозволяє реалізувати багаторівневий захист, розширену систему керування ключами або шифрування великих обсягів даних у реальному часі. Таким чином, критерій обмеження по ресурсах включає в себе аналіз енергоспоживання, вимог до пам’яті та процесора, а також можливість адаптації алгоритмів до умов, у яких вони будуть функціонувати. Ігнорування цього аспекту може призвести до збоїв, уповільненої роботи або надмірного навантаження на систему, що є неприйнятним для багатьох сфер, зокрема в медичних приладах, бездротових сенсорах, автомобільних системах або мобільних додатках.

Критерій «Тип даних, що шифруються» охоплює не лише структуру інформації, а й її обсяг, характер обробки та вимоги до продуктивності системи, що безпосередньо впливає на вибір алгоритму шифрування. При малому обсязі даних, до 10 МБ, наприклад, при передачі паролів, ключів, коротких повідомлень або невеликих конфігураційних файлів, пріоритетом стає мінімізація затримок і забезпечення високої швидкості обробки. У таких випадках найбільш ефективними є легкі симетричні алгоритми, що не потребують великих ресурсів і забезпечують надійний захист при малому навантаженні на систему. При середньому обсязі даних – від 10 до 500 МБ – наприклад, при захисті офісних документів, резервних копій користувача або архівів з даними, зростає значення балансу між швидкодією та рівнем безпеки. Алгоритм має бути здатним обробити значний обсяг без суттєвих затримок, водночас забезпечуючи достатній рівень стійкості до атак. У

таких випадках можуть застосовуватись як оптимізовані симетричні методи, так і гібридні схеми, у яких асиметричне шифрування використовується для захисту ключів, а основні дані шифруються симетричним алгоритмом. При великому обсязі даних – від 500 МБ до 10 ГБ – мова йде про захист повнорозмірних баз даних, відеофайлів у високій якості або серверних архівів. Тут критичними стають ефективність алгоритму в умовах навантаження, підтримка багатопотокової обробки, сумісність із файловими системами та можливість використання апаратного прискорення. Обираючи алгоритм, потрібно враховувати його здатність масштабуватись, уникати втрат продуктивності та забезпечувати захист даних протягом тривалого часу, включаючи збереження цілісності при передачі або зберіганні. Дуже великий обсяг даних – понад 10 ГБ – характерний для хмарних сервісів, серверів резервного копіювання, потокової архівації великих інформаційних масивів або систем обробки медіаконтенту. В таких умовах особливо важливим стає використання ефективних режимів шифрування, здатних працювати із частинами даних або блоками, уникати надлишкового споживання ресурсів і підтримувати високу пропускну здатність. Тут зазвичай застосовуються спеціальні гібридні або потокові рішення з використанням попереднього буферування, апаратного шифрування або розподілених обчислень. Окремою категорією є потокові дані, які передаються або обробляються в реальному часі – до них належать відео та аудіо трансляції, живі сенсорні потоки з IoT-пристроїв та інші динамічні формати. У таких випадках ключовими є мінімізація затримки, забезпечення стабільної якості передачі та захист від підміни або повторного відтворення пакетів. Алгоритми для такого типу даних мають бути швидкими, легкими, підтримувати потоковий режим та інтегруватись із транспортними протоколами. Шифрування тут виконується без попереднього накопичення даних, часто – блоками або фрагментами, що потребує високої оптимізації.

Критерій «Тип шифрування» визначає, яка криптографічна модель використовується для забезпечення конфіденційності та цілісності даних, і є ключовим фактором при виборі алгоритму в залежності від конкретного застосування або вимог користувача. У деяких випадках система або користувач

можуть вимагати застосування виключно симетричного або асиметричного шифрування через специфіку середовища, політики безпеки або архітектури програми. Симетричне шифрування, засноване на використанні одного спільного ключа для шифрування і дешифрування, зазвичай обирається у випадках, коли потрібна висока швидкість, обмежені ресурси або коли вже налагоджено безпечний канал для обміну ключами. Його часто використовують у вбудованих пристроях, при зберіганні даних на диску, у VPN-тунелях або для шифрування баз даних. Асиметричне шифрування, що базується на парі відкритого і закритого ключів, забезпечує зручний механізм для захисту при передачі даних між невідомими сторонами, особливо коли ключі попередньо не були обміняні. Такий тип шифрування широко використовується в електронному підписі, обміні ключами в мережах, при автентифікації користувачів або в електронній пошті. У випадках, коли тип шифрування не є принципово важливим для користувача, вибір здійснюється автоматично самою системою, яка оцінює ресурси пристрою, тип даних, необхідну швидкодію або інші параметри й пропонує найоптимальніший варіант – зазвичай це гібридна модель, що поєднує переваги обох підходів. Така гнучкість дозволяє зберігати баланс між безпекою та продуктивністю, не ускладнюючи налаштувань для кінцевого користувача. Отже, врахування типу шифрування як критерію дає змогу точніше адаптувати алгоритм до практичних потреб і умов використання.

Критерій «Сумісність зі стандартами» є важливим аспектом при виборі алгоритму шифрування, особливо в контексті застосування в державних установах, фінансових структурах, медичних інформаційних системах або інших сферах, де діють суворі нормативні вимоги до обробки даних. У таких випадках необхідна відповідність міжнародним або галузевим стандартам безпеки, як-от FIPS (Federal Information Processing Standards), або інші подібні документи, що регламентують допустимі криптографічні протоколи, формати ключів, довжини ключів, методи генерації випадкових чисел та інші параметри. Такі стандарти гарантують, що обраний метод шифрування є перевіреним, має документовану ефективність і визнаний у міжнародному середовищі як безпечний. З іншого боку, у деяких

проектах, дослідницьких ініціативах або внутрішніх системах, де немає вимог щодо сертифікації, можна застосовувати нестандартні або навіть експериментальні алгоритми шифрування. Це дозволяє тестувати нові криптографічні підходи, підвищувати продуктивність, адаптувати методи під конкретні умови чи пристрої, або ж зменшити навантаження на ресурси. Проте використання нестандартних методів тягне за собою ризики, пов'язані з недостатньою перевіркою надійності таких рішень, відсутністю підтримки у сторонніх системах та неможливістю проходження офіційної сертифікації безпеки. У контексті розробки системи автоматичного підбору шифрування важливо враховувати, чи користувач вимагає сувору відповідність стандартам, чи допускає можливість використання нестандартних алгоритмів – від цього безпосередньо залежатиме перелік доступних криптографічних рішень і варіантів реалізації.

Критерій «Масштабованість/підтримка паралельності» відіграє суттєву роль у виборі алгоритму шифрування, особливо коли йдеться про обробку великих обсягів даних або про реалізацію на високопродуктивних обчислювальних системах, таких як серверні кластери, графічні процесори чи хмарні платформи. У таких випадках важливо, щоб обраний алгоритм міг ефективно масштабуватись, тобто підтримував паралельну обробку блоків даних, що значно прискорює процес шифрування або дешифрування при розподіленні навантаження між кількома ядрами процесора або між різними вузлами системи. Наприклад, алгоритми, які працюють у блочному режимі з незалежними блоками, краще пристосовані до розподіленої обробки. Це дозволяє забезпечити високу продуктивність при роботі з потоковими даними, архівами, великими базами даних або у випадках, коли шифрування відбувається в реальному часі. Однак у деяких випадках масштабованість не є критичною – наприклад, при шифруванні невеликих обсягів інформації на кінцевих пристроях, де основний акцент робиться на мінімальному енергоспоживанні або простоті реалізації. У такому разі алгоритм може бути простим, лінійним і не підтримувати паралельність, що цілком прийнятно для мобільних або вбудованих систем. Таким чином, залежно від типу задачі й доступної обчислювальної інфраструктури, система має враховувати необхідність

підтримки масштабованості – це дозволить обирати між алгоритмами, орієнтованими на високу продуктивність, і тими, які підходять для обмежених умов.

Критерій «Підтримка автентифікації» визначає, наскільки важливо для користувача не лише зашифрувати дані, а й забезпечити їхню цілісність та підтвердити справжність джерела. У сучасних системах захисту даних все частіше постає вимога не просто приховати зміст повідомлення від сторонніх осіб, а й гарантувати, що дані не були змінені під час передавання або зберігання. Для цього використовуються алгоритми, які підтримують шифрування з автентифікацією. Такі методи одночасно забезпечують конфіденційність, цілісність і автентичність повідомлення, що є особливо важливим для захищених комунікацій, банківських операцій, обміну файлами в корпоративному середовищі тощо. Якщо ж підтвердження цілісності не є критично важливим для конкретного випадку – наприклад, якщо це внутрішнє шифрування даних, які не передаються по відкритих каналах або додатково контролюються іншими механізмами безпеки – тоді можливе використання звичайного шифрування без вбудованої автентифікації. Таким чином, залежно від вимог до захисту та контексту застосування, підтримка автентифікації може бути обов’язковим або опціональним параметром при виборі методу шифрування.

Критерій «Стійкість до помилок» визначає, наскільки алгоритм шифрування здатний витримати можливі помилки в процесі обробки даних без значних негативних наслідків для цілісності або безпеки зашифрованої інформації. Висока стійкість до помилок означає, що навіть при виникненні однієї помилки, наприклад, через пошкодження одного бітового чи блокового елемента, вся зашифрована інформація не буде порушена, і помилка не вплине на весь файл чи повідомлення. Це особливо важливо для середовищ, де передача даних може бути ненадійною або де важливо забезпечити надійність в умовах, коли можливі пошкодження, наприклад, при передачі даних через незахищені або шумні канали зв’язку. Алгоритми з високою стійкістю до помилок дозволяють уникнути ситуацій, коли одна помилка може призвести до повної втрати інформації або

серйозних помилок в обробці. Середня стійкість до помилок означає, що помилки можуть впливати на певну частину даних, але не на всю інформацію в цілому. Це може бути прийнятно в ситуаціях, коли пошкодження даних не є критичним, і система здатна відновити більшість інформації. Для деяких додатків, де не потрібна надвисока надійність, стійкість до помилок може бути менш важливою, оскільки втрата частини даних не призведе до серйозних наслідків. У деяких випадках, наприклад, при шифруванні малих обсягів даних чи у середовищах з високим рівнем контролю, проблема стійкості до помилок може бути неважливою, оскільки ймовірність пошкодження або помилки є дуже низькою. Однак для більш складних і критичних систем, що працюють з великими обсягами даних, важливо вибирати алгоритми, які забезпечують високу стійкість до помилок, щоб мінімізувати ризик втрати важливої інформації через технічні несправності.

У процесі вибору оптимального методу шифрування важливо враховувати цілу низку критеріїв, які взаємно впливають на ефективність захисту даних. Кожен критерій, будь то рівень безпеки, швидкодія, обмеження по ресурсах чи сумісність зі стандартами, має значення в залежності від умов застосування системи та типу даних, що шифруються. Зокрема, для різних платформ, таких як мобільні пристрої, веб-додатки або серверні системи, важливість кожного з критеріїв може змінюватися. Водночас, обмеження по ресурсах, таких як енергоспоживання чи обсяг пам'яті, можуть стати критичними факторами, особливо для вбудованих систем або IoT пристроїв. Ключовим є правильний баланс між усіма аспектами – від швидкості шифрування до підтримки автентифікації та стійкості до помилок. Ретельно вибраний метод шифрування здатен забезпечити надійний захист даних, зважаючи на специфіку застосування, й водночас ефективно функціонувати в заданих умовах. Тому детальний аналіз кожного з критеріїв та їх впливу на вибір алгоритму дозволяє зробити більш обґрунтоване рішення, яке задовольнить вимоги щодо безпеки та продуктивності.

1.6 Актуальні загрози безпеці та проблеми вибору оптимального методу шифрування

Актуальні загрози безпеці в інформаційних системах стають все більш складними та різноманітними, що робить вибір оптимального методу шифрування критично важливим для забезпечення захисту даних. В умовах постійного розвитку технологій, зловмисники постійно вдосконалюють свої методи атак, і це створює нові виклики для криптографії.

Основні загрози та вразливості в безпеці шифрування безпосередньо пов'язані з розвитком технологій та появою нових методів атак, які можуть скомпрометувати навіть найбільш надійні криптографічні системи. Однією з найбільших загроз є атаки на алгоритми шифрування, зокрема, спроби їх зламати за допомогою брутфорсу або математичних методів. І хоча сучасні криптографічні алгоритми є надзвичайно стійкими до таких атак, з розвитком обчислювальних потужностей і з використанням спеціалізованих апаратних засобів, терміни, за які можна провести атакуючи операції, поступово зменшуються. Наприклад, навіть найсучасніші алгоритми з довгими ключами можуть стати вразливими до атак в умовах доступу до квантових комп'ютерів, здатних швидко вирішувати певні математичні задачі, на яких ґрунтуються багато сучасних методів шифрування.

Іншою великою загрозою є проблема зберігання та управління криптографічними ключами. Незважаючи на те, що сам алгоритм може бути надзвичайно стійким, компрометація ключа може призвести до розкриття всіх зашифрованих даних. Використання слабких або легко доступних для зловмисників місць зберігання ключів, таких як незахищені сервери або недостатньо захищені пристрої, є частою причиною успішних атак. Відсутність належного управління ключами, коли вони зберігаються без достатнього рівня захисту, може стати фатальною вразливістю для всієї криптографічної системи.

Не менш важливою є вразливість до атак з вибору тексту або атаки на канали зв'язку, коли зловмисник має можливість отримати доступ до деякої кількості зашифрованих даних. Такі атаки дозволяють проаналізувати структуру шифрування та за допомогою статистичного аналізу чи інших методів отримати

інформацію про ключі або про самі дані. У випадку асиметричного шифрування одна з основних загроз – це атаки на приватні ключі, які можуть бути скомпрометовані через ненадійне зберігання або через поширення шкідливого програмного забезпечення. Наприклад, якщо приватний ключ потрапить до рук зловмисника, він зможе дешифрувати повідомлення, підписувати їх від імені власника ключа або викрасти чутливу інформацію.

Важливою загрозою є також слабкість реалізації криптографічних протоколів. Навіть при використанні правильних алгоритмів шифрування, погана реалізація протоколів може призвести до вразливостей. Це може бути спричинено помилками в коді програмного забезпечення, недоліками в апаратних засобах або вразливими точками в обміні даними. Наприклад, існують відомі випадки, коли невірно реалізовані криптографічні алгоритми могли бути обчислювані за допомогою відомих математичних методів, які не були враховані при розробці системи, що значно знижує рівень її безпеки.

Ще однією серйозною загрозою є атаки на канали передачі даних. Шифрування може бути ефективним лише в тому випадку, якщо захищені не лише дані на носіях, але й самі канали зв'язку. Якщо вразливість міститься в каналі передачі даних (наприклад, використання незахищених або відкритих мереж), зловмисник може здійснити атаку «man-in-the-middle», підмінивши або змінюючи зашифровані дані без відома відправника та отримувача.

Відсутність належного оновлення та підтримки криптографічних систем також є серйозною вразливістю. Технології постійно еволюціонують, і алгоритми, які колись вважалися надзвичайно надійними, можуть ставати вразливими через появу нових методів атак чи вдосконалення обчислювальних можливостей. Відсутність оновлень може призвести до використання застарілих або недостатньо захищених алгоритмів, що підвищує ризик злому даних.

Загалом, загрози та вразливості в безпеці шифрування можуть варіюватися від атак на алгоритми до людських помилок і недоліків у реалізації. Тому, для того щоб забезпечити ефективний захист даних, необхідно постійно удосконалювати криптографічні системи, контролювати зберігання ключів, використовувати

сучасні методи захисту каналів передачі даних, регулярно оновлювати використовувані алгоритми та протоколи, а також вибирати оптимальний метод шифрування для кожного випадку, щоб мінімізувати вразливості.

Вибір оптимального методу шифрування є складним завданням, яке включає не тільки технічні, а й стратегічні аспекти, що мають прямий вплив на рівень безпеки даних. Однією з основних проблем при виборі методу шифрування є необхідність балансування між швидкістю виконання та рівнем безпеки. Водночас, високий рівень безпеки не завжди гарантовано співвідноситься з високою швидкістю роботи алгоритму. Наприклад, деякі сучасні алгоритми, які пропонують надзвичайно високий рівень захисту, можуть бути дуже ресурсоємними, що призводить до збільшення часу на шифрування та розшифрування даних. Це особливо критично для систем з обмеженими ресурсами, таких як мобільні пристрої або вбудовані системи, де швидкість обробки даних має першочергове значення. Вибір оптимального методу вимагає врахування компромісу між цими двома параметрами. Також серед проблем є вибір між симетричними та асиметричними методами шифрування. Симетричні методи часто використовуються завдяки їхній швидкості та ефективності при роботі з великими обсягами даних. Проте основним їхнім недоліком є необхідність обміну ключами між відправником і отримувачем. Це створює додаткові ризики, особливо якщо ключі не захищені належним чином. З іншого боку, асиметричні методи забезпечують більшу безпеку завдяки використанню пари ключів – публічного та приватного, що дозволяє уникнути проблем з обміном ключами. Однак їхня головна проблема – значно більша повільність у порівнянні з симетричними методами. Тому вибір між цими методами залежить від специфіки завдання та вимог до безпеки і продуктивності.

Не менш важливим аспектом є також вибір методу шифрування в залежності від типу даних, які необхідно захистити. Для чутливих особистих даних, таких як паролі, фінансова інформація або медичні записи, часто вибираються методи з високим рівнем безпеки, навіть якщо це може супроводжуватися певними витратами на продуктивність. Для загальних даних або у випадках, коли висока

швидкість передачі важливіша за рівень безпеки, вибираються методи з меншою складністю, які все одно забезпечують достатній рівень захисту. Проблема полягає в тому, що немає універсального методу, який би підходив для всіх сценаріїв. Кожен випадок вимагає індивідуального підходу і вибору методів шифрування, що найкраще відповідають конкретним вимогам.

Іншою проблемою є підтримка та оновлення алгоритмів шифрування. Нові вразливості та атаки постійно з'являються, і криптографічні алгоритми, які раніше вважалися надійними, можуть стати уразливими. Це означає, що методи шифрування потребують регулярного аналізу та оновлення для забезпечення їхньої ефективності. Вибір старих, вже застарілих алгоритмів може призвести до серйозних проблем із безпекою, навіть якщо вони були достатньо надійними на момент їхнього впровадження. Це створює додаткові труднощі для організацій, які повинні постійно оновлювати свої системи шифрування та переконуватися, що використовувані алгоритми залишаються ефективними у боротьбі з новими загрозами. Окрім того, не можна забувати про вплив законодавчих обмежень і регулювань на вибір методу шифрування. У різних країнах можуть бути різні вимоги до шифрування, зокрема щодо того, які методи є дозволеними для використання. У деяких країнах існують обмеження на використання певних типів криптографії або на експорт шифрувальних технологій. Ці фактори можуть суттєво обмежити вибір методів шифрування, особливо для компаній, які працюють на міжнародному рівні.

Нарешті, проблема вибору оптимального методу шифрування також стосується людського фактора. Вибір і налаштування криптографічних алгоритмів вимагає високої кваліфікації від фахівців, а будь-яка помилка в налаштуваннях або в реалізації алгоритму може призвести до серйозних вразливостей. Часто виникає ситуація, коли організації вибирають найпопулярніші або найдоступніші методи шифрування без належної оцінки їхніх можливих недоліків або вимог до ресурсів, що також може призвести до компрометації безпеки.

Вибір оптимального методу шифрування є багатогранним завданням, яке включає численні фактори – від балансу між безпекою і швидкістю до врахування

специфіки даних, законодавчих обмежень і необхідності регулярного оновлення криптографічних систем. Тому важливо мати системний підхід до цього питання і орієнтуватися на специфічні потреби кожного конкретного випадку.

РОЗДІЛ 2 ВИЗНАЧЕННЯ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СИСТЕМИ ОЦІНКИ ЕФЕКТИВНОСТІ МЕТОДІВ ШИФРУВАННЯ

2.1 Аналіз вимог до системи

Система оцінки ефективності методів шифрування повинна забезпечувати користувача зручним і гнучким інструментом для вибору оптимального криптографічного алгоритму залежно від конкретних умов використання. Основне завдання полягає у тому, щоб на основі заданих параметрів – таких як рівень безпеки, вимоги до швидкодії, тип платформи, обмеження по ресурсах, характер і обсяг даних тощо – надати обґрунтовану рекомендацію щодо вибору шифрування, яке найкраще відповідає цим умовам.

Користувач взаємодіє з системою через веб-інтерфейс, де послідовно обирає значення критеріїв, що описують його потреби. Далі ці дані передаються на обробку машинній моделі, яка, проаналізувавши їх, визначає найбільш придатний метод шифрування. Система не просто зіставляє введені параметри з фіксованими правилами, а використовує модель машинного навчання, яка здатна знаходити залежності між умовами і характеристиками різних криптографічних алгоритмів, враховуючи множину тестових сценаріїв і результатів попереднього навчання.

Система повинна бути побудована з урахуванням модульності та розширюваності. Це забезпечує можливість у майбутньому додавати нові криптографічні алгоритми, змінювати або доповнювати перелік критеріїв вибору, а також адаптувати логіку обробки запитів до нових технічних вимог. Особлива увага приділяється продуктивності: система має швидко реагувати на запити користувача, працювати стабільно, а також забезпечувати високу точність у виборі відповідного методу шифрування відповідно до заданих параметрів.

2.2. Обґрунтування вибору архітектури клієнт-сервер

У контексті розробки системи оцінки ефективності методів шифрування вибір архітектури клієнт-сервер, що зображена на рис. 2.1, є цілком обґрунтованим і логічним, оскільки вона дозволяє чітко розмежувати функціональні ролі між

клієнтською частиною, яка відповідає за взаємодію з користувачем, і серверною частиною, яка реалізує основну логіку обробки даних, зберігання інформації та виконання машинного аналізу. Такий підхід забезпечує централізовану обробку запитів, що, у свою чергу, спрощує контроль за безпекою, підтримку коду та масштабування системи.

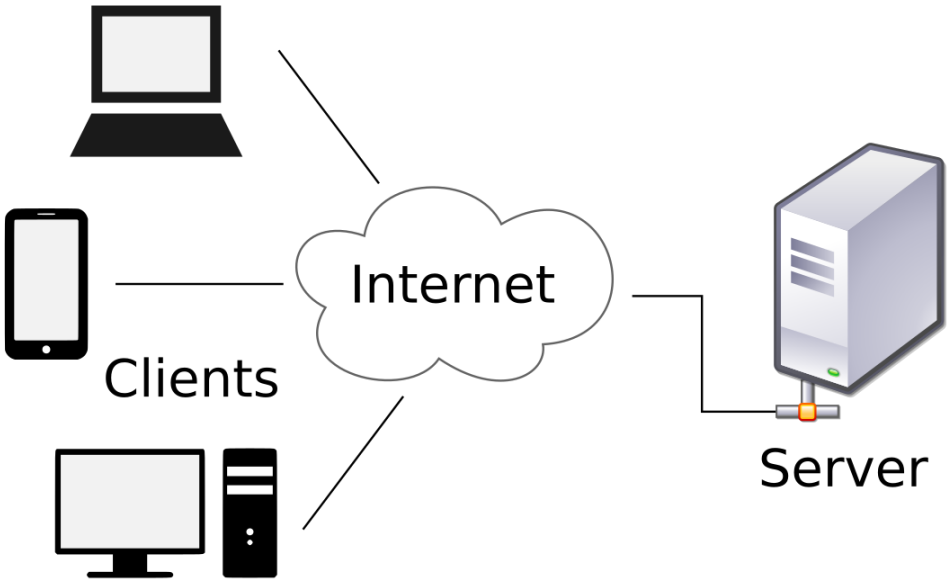


Рисунок 2.1 – Клієнт-серверна архітектура

Архітектура клієнт-сервер особливо доцільна у випадках, коли система має працювати з великою кількістю параметрів і обчислювальних сценаріїв, як у випадку з аналізом ефективності шифрувальних алгоритмів. Серверна частина несе на собі основне навантаження, виконуючи класифікацію на основі машинного навчання та надаючи користувачеві готові результати. Клієнт, зі свого боку, виконує роль інтерфейсу введення й відображення результатів, що дозволяє зменшити вимоги до ресурсів на боці користувача та підтримувати сумісність із різними пристроями, зокрема мобільними. Крім того, клієнт-серверна архітектура сприяє безперервному оновленню та вдосконаленню системи без необхідності втручання з боку користувача. Усі зміни, що стосуються логіки моделі, алгоритмів або бази знань, виконуються виключно на сервері, що забезпечує централізоване

управління. Це особливо важливо у випадку роботи з криптографією, де необхідна постійна актуалізація методів і відповідність сучасним вимогам.

Клієнт-серверна архітектура має низку важливих переваг, які роблять її особливо ефективною для реалізації систем. По-перше, вона дозволяє централізовано управляти логікою і обробкою даних на сервері, що спрощує підтримку, оновлення та масштабування системи без необхідності втручання з боку користувача. Це значно підвищує гнучкість розробки та експлуатації, адже всі зміни впроваджуються лише на сервері, а клієнти автоматично отримують актуальний функціонал без додаткових налаштувань. По-друге, розподіл обов'язків між клієнтом і сервером дозволяє оптимізувати використання ресурсів: складні обчислення і робота з великими даними виконуються на більш потужному сервері, тоді як клієнтська частина забезпечує зручний і легкий інтерфейс для взаємодії з користувачем. Це особливо важливо для забезпечення швидкої реакції системи та комфортного користувацького досвіду на різних пристроях, включно з мобільними. Крім того, клієнт-серверна модель підвищує безпеку, адже всі критичні операції і обробка конфіденційних даних відбуваються на сервері, що зменшує ризики компрометації даних на стороні користувача [13].

Завдяки цим властивостям клієнт-серверна архітектура є надійною, гнучкою та масштабованою платформою для побудови складних інформаційних систем, що потребують швидкої і точної обробки запитів, зручною взаємодії з користувачем та простого адміністрування. Саме тому вона є оптимальним вибором для системи оцінки ефективності методів шифрування.

2.3 Стек технологій для реалізації системи

2.3.1 Технології клієнтської частини

Клієнтська частина системи є першою точкою взаємодії користувача з додатком, тому важливо забезпечити її зручність, зрозумілість та привабливий зовнішній вигляд. Для реалізації інтерфейсу користувача було обрано технології HTML та CSS, що зображені на рис. 2.2, які є стандартом у розробці веб-застосунків. HTML відповідає за структуру веб-сторінок, дозволяючи визначати

розташування основних елементів, таких як форми, кнопки, таблиці, текстові блоки та інші елементи управління. Він створює каркас, на основі якого будуються всі візуальні компоненти системи. CSS буде використовуватись для стилізації HTML-елементів, надаючи можливість задавати кольори, шрифти, відступи, анімації, адаптивність до різних розмірів екранів та інші аспекти дизайну.

Рисунок 2.2 – Засоби розробки клієнтської частини

Для прискорення та спрощення розробки адаптивного і сучасного інтерфейсу також буде використано фреймворк Bootstrap 5, який забезпечує готові компоненти, сіткову систему та інструменти для створення інтуїтивного і узгодженого зовнішнього вигляду сторінок. За допомогою CSS і Bootstrap 5 інтерфейс стане не лише естетично привабливим, але й зручним у використанні на різних пристроях – від комп’ютерів до мобільних телефонів. Такий підхід дозволяє створити легку, швидку у завантаженні та інтуїтивно зрозумілу клієнтську частину, яка забезпечить позитивний досвід користувача під час взаємодії з системою вибору шифрування.

2.3.2 Технології серверної частини

Для реалізації серверної частини системи буде використано мову програмування C# у поєднанні з фреймворком ASP.NET MVC, зображеного на рис. 2.3, що забезпечить надійну, масштабовану та структуровано організовану архітектуру веб-застосунку. Однією з ключових причин вибору C# стало його безшовне поєднання з екосистемою .NET, зокрема фреймворком ASP.NET MVC,

який забезпечує розділення логіки програми, її інтерфейсу та взаємодії з даними. Це дозволяє краще організувати код, спрощує його підтримку та масштабування у майбутньому. Завдяки строгій структурі та вбудованим механізмам безпеки, C# сприяє зменшенню ризику помилок під час розробки та забезпечує високу стабільність виконання коду на сервері. Крім того, важливою перевагою C# є його широка підтримка сучасних протоколів, а також зручна інтеграція зі сторонніми сервісами, включаючи бібліотеки машинного навчання, бази даних та хмарні платформи. C# активно використовується у корпоративному середовищі, що підтверджує його ефективність у проєктах, які потребують високої надійності, продуктивності та безпеки. Завдяки цим перевагам C# є оптимальним вибором для побудови серверної частини системи, що забезпечує обробку користувацьких запитів, оцінку критеріїв та вибір найефективнішого алгоритму шифрування на основі результатів моделі машинного навчання.

ASP.NET MVC – це фреймворк від Microsoft, який дозволяє створювати динамічні веб-додатки з чітким розділенням логіки представлення, обробки запитів і доступу до даних. Він ґрунтується на шаблоні проектування MVC, який дозволяє структурувати код так, щоб він був максимально зрозумілим, підтримуваним і придатним до тестування. Крім того, ASP.NET MVC надає високу продуктивність, гнучкість у конфігурації маршрутизації, просту інтеграцію з JavaScript, HTML, CSS, а також можливість легко підключати зовнішні бібліотеки й API. Завдяки відсутності ViewState, зменшується навантаження на мережу та підвищується швидкодія [15]. Окрім цього, використовується чітке розділення відповідальностей між компонентами: модель відповідає за обробку даних і бізнес-логіку, представлення – за виведення інформації користувачу, а контролер – за обробку запитів та взаємодію між частинами системи. Такий підхід спрощує розробку, тестування та подальше розширення застосунку. Фреймворк підтримує маршрутизацію URL-адрес, що дозволяє створювати зручну структуру сторінок і покращує SEO. Також ASP.NET MVC архітектура, яка зображена на рис. 2.3, добре інтегрується з різноманітними базами даних і підтримує використання ORM-технологій, таких як Entity Framework, що спрощує роботу з даними. Його висока

продуктивність, безпека, надійність і підтримка сучасних засобів розробки роблять його ефективним рішенням для створення масштабованих, динамічних і захищених веб-систем.

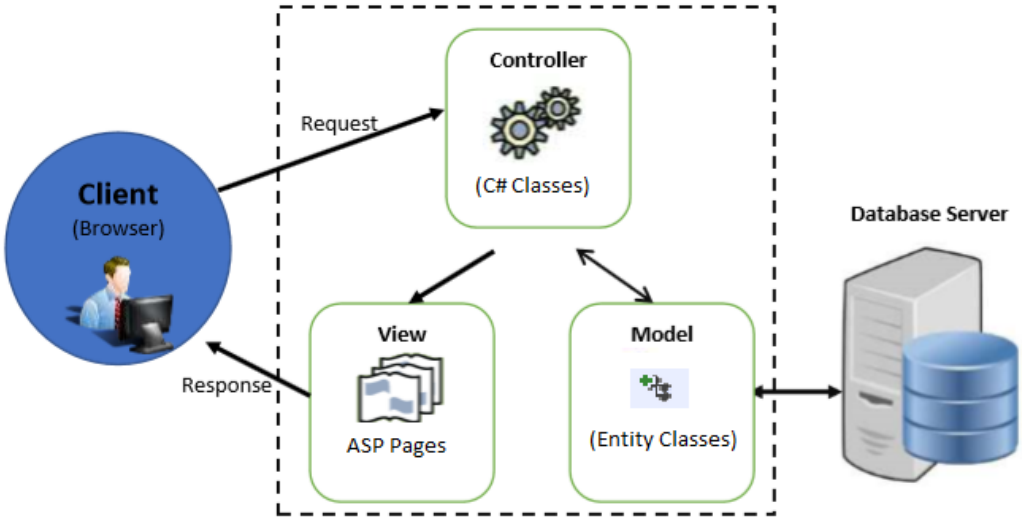


Рисунок 2.3 – MVC архітектура

2.4 Використання машинного навчання для автоматичного вибору методу шифрування

У сучасному цифровому середовищі, де обсяги даних невпинно зростають, а вимоги до безпеки стають дедалі суворішими, виникає необхідність у розумних і гнучких підходах до забезпечення конфіденційності інформації. Один із перспективних напрямів розвитку у цій сфері – використання машинного навчання для автоматичного вибору оптимального методу шифрування. Класичні підходи до вибору алгоритму зазвичай ґрунтуються на заздалегідь визначених параметрах або ручному аналізі ситуації, що не завжди є ефективним або достатньо гнучким в умовах динамічного цифрового середовища. Водночас машинне навчання дає змогу створити системи, які здатні самостійно аналізувати характеристики даних, умови обчислень, вимоги до безпеки та інші фактори, щоб зробити обґрунтований і адаптивний вибір алгоритму шифрування. Це дозволяє не лише підвищити

загальну ефективність системи захисту, а й забезпечити її масштабованість, стабільність та швидке реагування на нові виклики в галузі кібербезпеки.

Використання моделей машинного навчання для автоматичного визначення ефективного методу шифрування має низку важливих переваг, які суттєво підвищують якість та гнучкість систем захисту інформації. Однією з головних переваг є здатність до адаптивного аналізу даних: модель може автоматично враховувати характеристики середовища, обмеження по ресурсах, тип даних, що шифруються, необхідність автентифікації або відповідність стандартам безпеки. Завдяки цьому система може динамічно змінювати або обирати алгоритм шифрування, який найбільш ефективно відповідає конкретній ситуації, без необхідності ручного втручання з боку користувача чи адміністратора. Ще однією важливою перевагою є здатність обробляти великі обсяги вхідних параметрів, що дозволяє враховувати складні залежності між різними критеріями, які можуть бути недоступними для класичних логічних правил [17]. Окрім цього, моделі машинного навчання можуть навчатися на основі історичних даних і вдосконалювати точність своїх рішень у процесі використання, що сприяє постійному підвищенню ефективності захисту. У випадках, коли з’являються нові загрози або змінюються вимоги до безпеки, модель може швидко адаптуватися, оновивши свою поведінку на основі нових даних. Використання ML-підходів дозволяє досягти більш високого рівня автоматизації, точності, швидкодії та надійності в процесі вибору шифрування, забезпечуючи захист, який відповідає сучасним викликам кіберпростору.

2.4.1 Використання ML.NET для автоматизації вибору шифрування

Модель машинного навчання для автоматичного вибору методу шифрування на мові програмування C# є актуальним підходом, особливо в умовах інтеграції з іншими компонентами програмного забезпечення на платформі .NET. Одним із найбільш зручних та ефективних інструментів для розробки ML-рішень у середовищі C# є бібліотека ML.NET – це фреймворк від Microsoft, який дозволяє створювати, тренувати та впроваджувати моделі машинного навчання без потреби

в знанні мов типу Python або R. ML.NET підтримує широкий спектр алгоритмів для класифікації, регресії, кластеризації, а також має засоби для обробки даних, побудови конвеєрів навчання, оптимізації гіперпараметрів та збереження моделей у вигляді файлів, які можна легко інтегрувати у великі додатки. Процес створення такої моделі передбачає кілька важливих етапів: на початку формуються вхідні дані, які мають містити опис параметрів задачі шифрування – наприклад, тип платформи, обсяг даних, потребу в автентифікації, обмеження по ресурсах тощо. Ці дані перетворюються в числові або категоріальні ознаки, які подаються на вхід моделі. Потім відбувається навчання на основі існуючої вибірки, де кожен запис містить відповідні параметри та найбільш доцільний тип шифрування [19]. Зазвичай для такого завдання обирається алгоритм багатокласової класифікації, наприклад, DecisionTree, LightGBM або FastTree, які підтримуються ML.NET і забезпечують хорошу продуктивність та точність. Після навчання модель може бути збережена у файл формату ZIP або BIN, а потім завантажена для використання в реальному часі.

Готову модель можна інтегрувати в додаток, що приймає параметри від користувача або системи і на їх основі в режимі реального часу визначає найбільш оптимальний метод шифрування. Це особливо зручно у випадках, коли система працює з великою кількістю даних або у різних середовищах, де параметри постійно змінюються. Використання ML.NET дозволяє уникнути надлишкових залежностей від зовнішніх ML-фреймворків, залишаючи весь стек технологій у межах екосистеми C#, що спрощує обслуговування, деплой та безпековий контроль.

2.4.2 Аналіз методів машинного навчання для задачі вибору шифрування

Для розв’язання задачі автоматичного вибору методу шифрування за допомогою машинного навчання можуть застосовуватися різні підходи, кожен із яких має свої особливості, сильні сторони та обмеження.

Класифікаційна модель навчання є одним із фундаментальних типів моделей машинного навчання, призначеним для вирішення задач, де необхідно віднести вхідні дані до однієї з кількох наперед визначених категорій. У контексті вибору методу шифрування така модель може аналізувати набір вхідних параметрів – наприклад, обсяг даних, тип платформи, необхідність автентифікації, обмеження по ресурсах або вимоги до безпеки – і на їх основі класифікувати, який саме алгоритм шифрування є найбільш придатним для конкретної ситуації. Процес побудови такої моделі починається зі збору навчальної вибірки, де кожен запис містить значення характеристик і відповідну мітку класу – тобто назву алгоритму шифрування, який виявився найефективнішим у подібних умовах. Далі ці дані проходять попередню обробку: нормалізацію числових значень, перетворення категоріальних ознак у числову форму, заповнення відсутніх значень тощо. Навчання класифікаційної моделі може базуватись на різних алгоритмах – до найпоширеніших належать дерева рішень, метод опорних векторів, логістична регресія, нейронні мережі, градієнтний бустинг, наївний баєсівський класифікатор та інші. Вибір алгоритму залежить від складності задачі, обсягу даних і вимог до точності та швидкодії. Після навчання модель починає «розуміти» закономірності між комбінаціями параметрів та відповідними рішеннями. Коли на вхід подається новий набір параметрів, модель аналізує їх, порівнює з тим, що вивчила раніше, і повертає прогноз – тобто назву методу шифрування, який найкраще відповідає цим умовам. Головною перевагою класифікаційної моделі є її здатність приймати автоматизовані рішення на основі досвіду, не потребуючи ручного налаштування кожного разу. Це дозволяє створювати інтелектуальні системи, які швидко адаптуються до змін, автоматично підлаштовуються під різні сценарії використання і забезпечують оптимальний вибір без участі людини [24]. У сфері інформаційної безпеки, де важливі як продуктивність, так і надійність, класифікаційна модель дозволяє досягти балансу між ефективністю алгоритму та специфікою умов його застосування.

Ще однією моделлю навчання є регресійна модель. Вона є типом моделі машинного навчання, що використовується для прогнозування числових значень

на основі вхідних даних. Відрізняється від класифікаційних моделей тим, що замість того, щоб відносити вхід до певної категорії або класу, регресія намагається передбачити конкретне значення. У контексті вибору методу шифрування регресійна модель може бути використана для прогнозування таких параметрів, як час виконання алгоритму шифрування, його енергоспоживання, рівень безпеки, а також інших числових характеристик, які необхідно оптимізувати в рамках обраного шифрування. Процес побудови регресійної моделі аналогічний класифікаційній: початково збираються дані, що містять вхідні параметри та відповідні числові значення, які можуть включати час виконання, споживання пам'яті, обсяг даних та інші метрики. Далі ці дані проходять етапи обробки, такі як нормалізація та усунення пропущених значень. Після цього вибирається алгоритм регресії, який може бути лінійним, поліноміальним, на основі методів опорних векторів, нейронних мереж або інших підходів. Модель тренується, щоб мінімізувати різницю між фактичними значеннями та прогнозованими, використовуючи методи оптимізації, такі як градієнтний спуск або інші техніки. Після навчання регресійна модель може прогнозувати числові значення на основі нових вхідних параметрів. Для вибору методу шифрування це може бути корисним, якщо необхідно оптимізувати не тільки вибір алгоритму, а й його характеристики в залежності від конкретних умов. Наприклад, модель може передбачити, який алгоритм буде найбільш ефективним з точки зору часу обробки або енергоспоживання при наявності певних обмежень по ресурсах. Однією з головних переваг регресійних моделей є здатність давати точні прогнози на основі числових даних, що дозволяє точно оцінити, який метод шифрування буде оптимальним в умовах конкретних вимог, таких як максимізація продуктивності чи зменшення часу виконання [25]. Проте основним викликом є потреба у великій кількості даних для побудови точних прогнозів і вразливість моделі до помилок у вхідних даних, що можуть негативно впливати на результати.

Ще один метод навчання – кластеризація. Він є одним з основних методів машинного навчання, що використовується для групування об'єктів або даних за схожістю без наявності міток або попередньо визначених класів. Цей метод

належить до задач неконтрольованого навчання, що означає, що алгоритм не має чітких вказівок щодо того, як правильно класифікувати дані. Кластеризація знаходить застосування в ситуаціях, коли необхідно групувати схожі за певними ознаками елементи в одну категорію без попереднього визначення цих груп. У контексті вибору методу шифрування кластеризація може бути використана для того, щоб розділити набір шифрувальних алгоритмів на кілька груп на основі схожих характеристик, таких як рівень безпеки, швидкість виконання, енергоспоживання або інші важливі параметри. Алгоритм кластеризації може автоматично визначити, які методи шифрування мають схожі властивості, що дозволяє системі швидко відносити нові методи або нові умови до певної категорії. Процес кластеризації зазвичай включає кілька етапів: на початку зібрані дані попередньо обробляються, що може включати нормалізацію або стандартизацію числових характеристик для уніфікації їх шкал. Далі вибирається відповідний алгоритм кластеризації, такий як K-середніх, ієрархічна кластеризація або методи на основі DBSCAN, в залежності від типу даних і вимог до результатів. Ці алгоритми працюють таким чином, що вони намагаються мінімізувати відстань між елементами всередині одного кластеру і максимізувати відстань між різними кластерами. Кластеризація дає змогу виявити структуру в даних, яка не була очевидною на перший погляд. Це особливо корисно в ситуаціях, коли немає чіткої мети або ж коли дані є занадто великими і складними для ручного аналізу. Для вибору методу шифрування кластеризація дозволяє групувати алгоритми за такими характеристиками, як тип застосовуваних ресурсів, час виконання або енергоспоживання, що дає змогу швидше знаходити відповідні рішення в різних сценаріях. Однак варто зазначити, що кластеризація не завжди може забезпечити чіткі або однозначні результати, оскільки визначення оптимальної кількості кластерів може бути складним завданням, а вибір алгоритму кластеризації залежить від конкретних характеристик даних. Крім того, кластеризація не дає точних прогнозів або рішень, а лише допомагає виявити структуру в даних, що потребує додаткової обробки для точного вибору методу шифрування.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ СИСТЕМИ ОЦІНКИ ЕФЕКТИВНОСТІ МЕТОДІВ ШИФРУВАННЯ

3.1 Загальна архітектура системи

Система оцінки ефективності методів шифрування побудована за принципом клієнт-серверної архітектури, що забезпечує гнучкість, масштабованість та можливість розширення функціоналу в майбутньому. Основною метою архітектури є інтеграція інтерфейсу для користувача, серверної логіки обробки запитів, моделі машинного навчання та алгоритмів оцінки шифрування в єдину узгоджену систему. Клієнтська частина реалізована з використанням технологій HTML, CSS та Razor (в рамках ASP.NET MVC), що дозволяє створити інтуїтивно зрозумілий інтерфейс, у якому користувач вводить вхідні параметри (наприклад, обсяг даних, критичність до продуктивності, необхідний рівень безпеки тощо). Запити з цих даних передаються до серверної частини через HTTP. Серверна частина реалізована на платформі ASP.NET MVC та відповідає за обробку запитів, взаємодію з ML-моделлю, виконання розрахунків та повернення результатів клієнту. Центральним компонентом серверної логіки є модуль аналізу вхідних критеріїв, який передає їх до моделі машинного навчання, створеної за допомогою ML.NET. Модель прогнозує оптимальний метод шифрування, виходячи з набору попередньо підготовлених даних і тренувального набору. ML-модель розміщується безпосередньо на сервері, що дозволяє виконувати прогноз у реальному часі без звернення до зовнішніх сервісів. Це гарантує швидкість обробки та зменшує залежність від сторонніх рішень.

Уся система побудована модульно, що дозволяє легко оновлювати окремі компоненти, додавати нові алгоритми шифрування, змінювати критерії оцінки або перенавчати модель без необхідності повної реконструкції програмного забезпечення. Загалом, архітектура системи забезпечує високу продуктивність, зручність використання та можливість адаптації до нових умов у сфері криптографії та інформаційної безпеки.

3.2 Розробка користувацького інтерфейсу

Користувацький інтерфейс є важливою складовою системи, оскільки забезпечує взаємодію кінцевого користувача з функціоналом оцінки та вибору методів шифрування. Основне завдання інтерфейсу – зробити процес введення параметрів максимально простим, зрозумілим і зручним для користувача, а також надати вичерпну та наочну інформацію про результати оцінки. Інтерфейс реалізований у рамках технології ASP.NET MVC із використанням Razor-шаблонів, HTML5, CSS3 та фреймворку Bootstrap 5. Завдяки цьому забезпечено чітке розділення логіки подання, обробки та виводу інформації. Дизайн інтерфейсу побудований відповідно до принципів адаптивності, що дозволяє використовувати систему як на настільних ПК, так і на мобільних пристроях. На головній сторінці користувачу пропонується блок для введення вхідних критеріїв, який проілюстрований на рис. 3.1, що впливає на вибір методу шифрування.

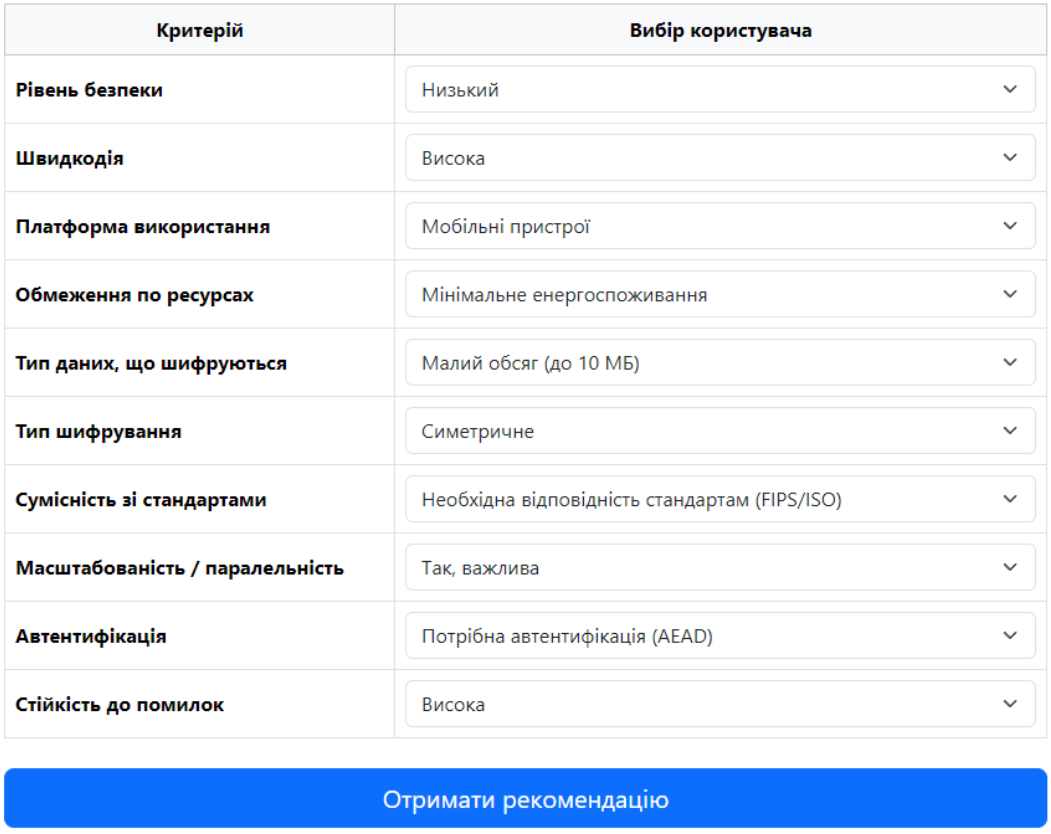


Рисунок 3.1 – Блок вводу критеріїв оцінки шифрування

Серед таких критеріїв – тип даних, їх обсяг, обмеження по швидкодії, бажаний рівень безпеки, платформа використання, та інші параметри, релевантні до конкретного сценарію. Кожен вхідний параметр супроводжується коротким описом в дужках, що допомагає уникнути помилок і підвищує зручність використання. Після заповнення форми користувач натискає кнопку «Отримати рекомендацію», після чого дані надсилаються на сервер для обробки. Результат відображається у вигляді блоку, який наведено на рис. 3.2, та містить назву рекомендованого методу шифрування.

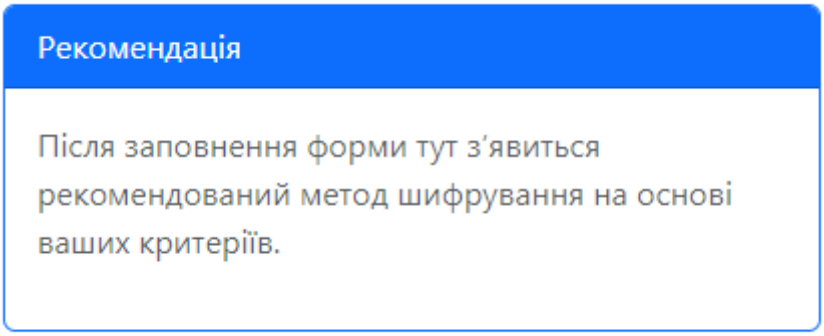


Рисунок 3.2 – Блок для виведення рекомендованого методу шифрування

Загалом, розроблений UI забезпечує простий і зрозумілий доступ до функцій системи, підтримує логічну послідовність дій користувача та сприяє прийняттю обґрунтованих рішень у виборі методу шифрування.

3.3 Логіка роботи моделі машинного навчання

У рамках розробки системи оцінки методів шифрування було створено модель машинного навчання, яка на основі заданих критеріїв визначає оптимальний алгоритм шифрування з чотирьох часто використовуваних: AES, RSA, ECC або ChaCha20. Модель реалізована з використанням бібліотеки Microsoft.ML у середовищі C# та інтегрована в додаток ASP.NET MVC, що дозволяє автоматизувати вибір криптографічного алгоритму для різних сценаріїв використання. Принцип роботи моделі охоплює кілька етапів: підготовку даних,

їхнє перетворення, навчання моделі, прогнозування результатів і оцінку ефективності. Логіка роботи моделі базується на багатокласовій класифікації, де вхідні дані, представлені категоріальними ознаками, використовуються для передбачення одного з чотирьох класів, що відповідають алгоритмам шифрування.

Вхідні дані для моделі представлені у вигляді CSV-файлу, який містить 11 колонок: 10 ознак, що описують критерії вибору алгоритму шифрування (SecurityLevel, Performance, Platform, ResourceLimits, DataType, EncryptionType, StandardCompliance, Scalability, Authentication, ErrorResistance), і одна цільова змінна (Algorithm). Ознаки є категоріальними, тобто мають дискретні значення, наприклад, SecurityLevel може бути High, Medium або Low, а Platform – Server, Web, Mobile або IoT. Цільова змінна Algorithm вказує на один із чотирьох алгоритмів шифрування. Для навчання моделі було використано 300 унікальних записів, створених із забезпеченням різноманітності комбінацій ознак і приблизного розподілу алгоритмів: 25% AES, 30% ChaCha20, 25% ECC і 20% RSA. Такий розподіл було обрано для відображення їхньої частоти використання в реальних сценаріях, з більшим акцентом на ChaCha20 через популярність у мобільних системах і меншим на RSA через специфічність асиметричного шифрування. Цей підхід дозволяє моделі навчатися на репрезентативному наборі даних, що відображає різні сценарії використання шифрування.

Оскільки алгоритми машинного навчання, як-от LightGBM, що використовується в моделі, працюють із числовими даними, категоріальні ознаки необхідно перетворити в числовий формат. Для цього застосовується метод OneHotHashEncoding, який є частиною пайплайну обробки даних у Microsoft.ML. Цей метод перетворює кожне категоріальне значення в набір бінарних ознак, використовуючи хешування для зменшення розмірності. Наприклад, для ознаки Platform із чотирма значеннями (Server, Web, Mobile, IoT) створюється набір бітів, де кожне значення кодується унікальним хешем. Після кодування всі ознаки об’єднуються в єдиний числовий вектор за допомогою операції Concatenate, який стає вхідним для алгоритму класифікації. Цільова змінна Algorithm також

кодується в числовий формат, де кожному алгоритму (AES, ChaCha20, ECC, RSA) присвоюється унікальний індекс (0, 1, 2, 3), що відповідає порядку класів у моделі.

Для вибору оптимального алгоритму класифікації використовується AutoML, інструмент автоматизованого машинного навчання від Microsoft.ML. AutoML проводить експеримент, тестуючи різні алгоритми багатокласової класифікації (LbfgsMaximumEntropy, LightGbm, FastTree) протягом заданого часу – у даному випадку 60 секунд. Критерієм вибору найкращого алгоритму є MacroAccuracy, метрика, яка оцінює середню точність передбачень для всіх класів, що особливо важливо для збалансованої оцінки в задачах із нерівномірним розподілом класів. У результаті експерименту AutoML обрав алгоритм LightGBM для багатокласової класифікації (LightGbmMulti), який є варіантом градієнтного бустингу, оптимізованим для роботи з табличними даними та категоріальними ознаками. LightGBM будує ансамбль дерев рішень, де кожне дерево послідовно коригує помилки попередніх, використовуючи градієнтний спуск для мінімізації функції втрат, у даному випадку – логарифмічних втрат (LogLoss) для багатокласової класифікації.

Процес навчання моделі починається з розбиття даних на тренувальну (80%) і тестову (20%) вибірки за допомогою методу TrainTestSplit. Таке співвідношення є стандартне для машинного навчання, щоб забезпечити достатній обсяг даних для навчання моделі LightGBM і надійну оцінку її точності на репрезентативній тестовій вибірці. Тренувальна вибірка використовується для побудови моделі, тоді як тестова – для оцінки її ефективності. Під час навчання LightGBM аналізує числові вектори ознак і відповідні мітки класів, оптимізуючи дерева рішень для максимізації точності передбачень. Кожне дерево в ансамблі оцінює, які значення ознак (наприклад, SecurityLevel = High, EncryptionType = Symmetric) найсильніше впливають на вибір певного алгоритму шифрування. Після навчання модель зберігається у файл encryption_model.zip, що дозволяє використовувати її для подальших передбачень у веб-додатку.

Для прогнозування оптимального алгоритму шифрування модель приймає набір критеріїв, представлених у вигляді об’єкта класу EncryptionCriteria. Ці

критерії проходять той самий процес перетворення: категоріальні значення кодуються за допомогою OneHotHashEncoding, об’єднуються в числовий вектор і подаються на вхід LightGBM. Модель видає два результати: передбачений клас (наприклад, AES) і масив ймовірностей (Score), де кожен елемент відповідає ймовірності належності до одного з чотирьох класів. Наприклад, для тестового прикладу з високим рівнем безпеки, симетричним шифруванням і великими даними модель передбачила AES із ймовірністю 95.07%, що свідчить про високу впевненість у рішенні.

Перетворення результату назад у категоріальний формат відбувається автоматично: числовий індекс класу (наприклад, 0) зіставляється з відповідним алгоритмом (AES) за допомогою внутрішнього словника класів, створеного під час навчання. Це дозволяє відобразити результат у зрозумілій формі для користувача, наприклад, у веб-інтерфейсі ASP.NET MVC. Для оцінки ефективності моделі використовуються метрики, такі як MacroAccuracy і LogLoss, які обчислюються на тестовій вибірці. MacroAccuracy показує середню точність передбачень для всіх класів, тоді як LogLoss оцінює, наскільки добре модель розподіляє ймовірності між класами – нижче значення LogLoss вказує на кращу якість передбачень.

Логіка вибору результату моделлю спирається на залежності, виявлені під час навчання. Наприклад, для сценаріїв із високим рівнем безпеки, симетричним шифруванням і серверною платформою модель частіше обирає AES, тоді як для мобільних пристроїв із низькими ресурсами та високою продуктивністю – ChaCha20. Ці закономірності відображають розподіл даних у навчальному наборі, де кожен алгоритм асоціюється з певними комбінаціями критеріїв. Завдяки LightGBM модель здатна виявляти складні нелінійні залежності між ознаками, що забезпечує високу точність передбачень навіть для нових, раніше не бачених комбінацій.

Таким чином, модель машинного навчання є ефективним інструментом для автоматизованого вибору алгоритмів шифрування. Вона поєднує потужність LightGBM для обробки категоріальних даних, автоматизацію AutoML для вибору оптимального алгоритму та гнучкість Microsoft.ML для інтеграції в додаток.

Перетворення даних у числовий формат і назад забезпечує коректну роботу моделі з категоріальними ознаками, а висока ймовірність правильних передбачень, як у тестовому прикладі, підтверджує її практичну цінність для вирішення задачі оцінки методів шифрування.

3.4 Розробка серверної частини

Серверна частина системи виконує ключову роль у процесі обробки запитів користувача, оцінювання вхідних критеріїв та визначення найоптимальнішого методу шифрування. Саме на цьому рівні реалізовано основну бізнес-логіку, зокрема формування моделей, обробку даних, взаємодію з компонентами машинного навчання та повернення результату до користувача. Одним із базових компонентів серверної частини є модель EncryptionCriteriaViewModel, що зображена на рис. 3.3, та використовується для збирання та зберігання критеріїв, що вводяться користувачем під час заповнення форми на вебсайті. Ці дані надалі передаються до контролера для подальшої обробки та аналізу.

```
namespace EncryptionScanner.ViewModels
{
    5 references
    public class EncryptionCriteriaViewModel
    {
        2 references
        public string SecurityLevel { get; set; } = default!;
        2 references
        public string Performance { get; set; } = default!;
        2 references
        public string Platform { get; set; } = default!;
        2 references
        public string ResourceLimits { get; set; } = default!;
        2 references
        public string DataType { get; set; } = default!;
        2 references
        public string EncryptionType { get; set; } = default!;
        2 references
        public string StandardCompliance { get; set; } = default!;
        2 references
        public string Scalability { get; set; } = default!;
        2 references
        public string Authentication { get; set; } = default!;
        2 references
        public string ErrorResistance { get; set; } = default!;
        3 references
        public string? Algorithm { get; set; }
    }
}
```

Рисунок 3.3 – Клас EncryptionCriteriaViewModel для представлення вхідних критеріїв шифрування

Клас EncryptionCriteriaViewModel, розташований у просторі імен EncryptionScanner.ViewModels, є частиною серверної логіки, яка відповідає за передачу даних між користувачем і системою. Він виконує роль моделі представлення (ViewModel), яка використовується для прийому вхідних параметрів, введених користувачем через вебінтерфейс, і подальшої їх обробки в контролері. Кожна властивість цього класу відповідає певному критерію, який враховується при виборі найбільш ефективного методу шифрування. Усі властивості є рядковими і оголошені як автоматичні, тобто мають get та set для зручного зчитування і запису значень. Для уникнення попереджень компілятора про відсутність ініціалізації рядкових non-nullable полів використано вираз = default!;, який гарантує, що значення будуть заповнені під час виконання, коли користувач надішле форму. Окрема властивість Algorithm є опціональною і зберігає результат – назву алгоритму шифрування, який система визначить як найбільш відповідний за введеними критеріями. Цей клас фактично виступає контейнером для даних, який бере участь у зв’язуванні форми на клієнтській частині з логікою серверної обробки, зокрема з ML-моделлю, яка на основі цих критеріїв прогнозує оптимальний метод шифрування.

Після того як визначено модель для збирання вхідних критеріїв шифрування, наступним кроком є організація обробки цих даних на рівні контролера, який виступає посередником між користувачем, логікою машинного навчання та представленням результату. Саме цей функціонал реалізовано у файлі HomeController, який наведено на рис. 3.4. Він обробляє запити з форми на головній сторінці та забезпечує виклик відповідної ML-моделі для формування рекомендацій.

```
namespace EncryptionScanner.Controllers
{
    1 reference
    public class HomeController : Controller
    {
        private readonly IMLService _mLService;

        0 references
        public HomeController(IMLService mLService)
        {
            _mLService = mLService;
        }

        [HttpGet]
        0 references
        public IActionResult Index()
        {
            return View();
        }

        [HttpPost]
        0 references
        public IActionResult Index(EncryptionCriteriaViewModel encryptionCriteria)
        {
            var result = _mLService.GetPrediction(encryptionCriteria);
            encryptionCriteria.Algorithm = result;
            return View(encryptionCriteria);
        }
    }
}
```

Рисунок 3.4 – Логіка обробки запитів у контролері HomeController

Контролер реалізує два методи: один для обробки HTTP-запитів типу GET, інший – POST. У методі Index() з атрибутом [HttpGet] просто повертається порожнє представлення (View), що відображає сторінку з формою для введення критеріїв. Цей метод викликається, коли користувач лише відкриває сторінку без надсилання даних. Основна логіка реалізована в перевантаженому методі Index(EncryptionCriteriaViewModel encryptionCriteria) з атрибутом [HttpPost]. Цей метод приймає об'єкт EncryptionCriteriaViewModel, який автоматично заповнюється даними, введеними користувачем у формі. Після цього контролер викликає метод GetPrediction з сервісу машинного навчання IMLService, передаючи йому отримані критерії. Сервіс обробляє ці дані, аналізує їх за допомогою попередньо навченої моделі ML.NET і повертає назву алгоритму шифрування, який є найбільш доцільним відповідно до заданих параметрів. Отриманий результат записується у властивість Algorithm об'єкта EncryptionCriteriaViewModel, після чого ця ж модель передається назад до представлення, вже з включеним результатом – тобто назвою рекомендованого методу шифрування.

Продовжуючи логіку обробки запитів, описану у файлі HomeController, наступним ключовим елементом системи є сервіс машинного навчання, який безпосередньо відповідає за прийняття рішення щодо вибору оптимального методу шифрування на основі заданих користувачем критеріїв. Уся ця функціональність реалізована у класі MLService, що є конкретною реалізацією інтерфейсу IMLService та наслідує базовий клас MLServiceBase, який зображено на рис. 3.5.

```
namespace EncryptionScanner.Services
{
    2 references
    public class MLService : MLServiceBase, IMLService
    {
        0 references
        public MLService(IOptions<MLModelData> options) : base(options)
        {

        }

        2 references
        public string GetPrediction(EncryptionCriteriaViewModel criteria)
        {
            var encryptionCriteria = new EncryptionCriteria
            {
                SecurityLevel = criteria.SecurityLevel,
                Performance = criteria.Performance,
                Platform = criteria.Platform,
                ResourceLimits = criteria.ResourceLimits,
                DataType = criteria.DataType,
                EncryptionType = criteria.EncryptionType,
                StandardCompliance = criteria.StandardCompliance,
                Scalability = criteria.Scalability,
                Authentication = criteria.Authentication,
                ErrorResistance = criteria.ErrorResistance
            };

            var result = Predict(encryptionCriteria);
            return result.Algorithm;
        }
    }
}
```

Рисунок 3.5 – Реалізація сервісу машинного навчання для вибору методу шифрування

Основна логіка зосереджена в методі GetPrediction, який приймає об'єкт EncryptionCriteriaViewModel, що надходить з контролера. У межах цього методу дані з ViewModel копіюються у внутрішню модель EncryptionCriteria. Це необхідно для сумісності з методами, які очікують конкретний тип об’єкта, що містить вхідні характеристики у відповідному форматі. Після створення об'єкта EncryptionCriteria викликається метод Predict, успадкований від базового класу MLServiceBase. Цей

метод передає зібрані параметри до навченої ML.NET-моделі, яка аналізує критерії та повертає об'єкт із передбаченням – тобто найдоцільнішим алгоритмом шифрування для заданого набору характеристик. Із результату витягується лише назва методу шифрування (result.Algorithm), яка і повертається назад у контролер для подальшого відображення користувачу.

Наступним важливим елементом є модель даних EncryptionCriteria, що наведена на рис. 3.6, і використовується як внутрішній тип для передачі вхідних параметрів у ML-модель. Цей клас чітко визначає структуру інформації, яка необхідна для оцінки та вибору оптимального методу шифрування.

```
namespace EncryptionScanner.Models
{
    4 references
    public class EncryptionCriteria
    {
        [LoadColumn(0)]
        1 reference
        public string SecurityLevel { get; set; }
        [LoadColumn(1)]
        1 reference
        public string Performance { get; set; }
        [LoadColumn(2)]
        1 reference
        public string Platform { get; set; }
        [LoadColumn(3)]
        1 reference
        public string ResourceLimits { get; set; }
        [LoadColumn(4)]
        1 reference
        public string DataType { get; set; }
        [LoadColumn(5)]
        1 reference
        public string EncryptionType { get; set; }
        [LoadColumn(6)]
        1 reference
        public string StandardCompliance { get; set; }
        [LoadColumn(7)]
        1 reference
        public string Scalability { get; set; }
        [LoadColumn(8)]
        1 reference
        public string Authentication { get; set; }
        [LoadColumn(9)]
        1 reference
        public string ErrorResistance { get; set; }
        [LoadColumn(10)]
        0 references
        public string Algorithm { get; set; }
    }
```

Рисунок 3.6 – Модель даних EncryptionCriteria

Кожна властивість класу відповідає конкретному критерію, що впливає на вибір алгоритму. Властивість Algorithm містить результат – рекомендований алгоритм шифрування, що є цільовою змінною для ML-моделі. Особливістю цього класу є використання атрибутів [LoadColumn(n)], які вказують на індекси колонок

у вхідних даних, що завантажуються в модель. Це забезпечує коректне зіставлення даних із CSV-файлу з відповідними властивостями класу.

Розглянемо реалізацію базового сервісу машинного навчання у файлі MLServiceBase, який наведено на рис. 3.7–3.8. Цей клас відповідає за завантаження, навчання, збереження та застосування ML-моделі для оцінки методів шифрування.

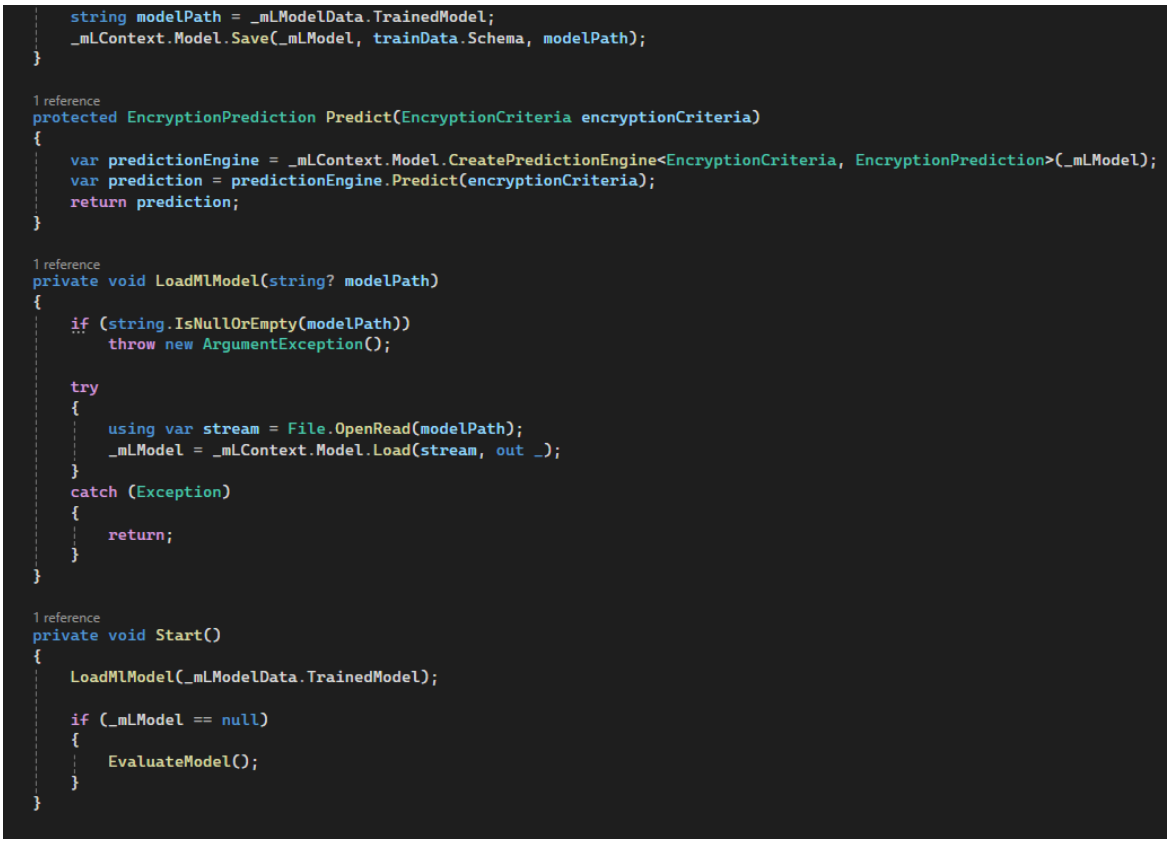
```
namespace EncryptionScanner.Services
{
    3 references
    public class MLServiceBase
    {
        private readonly MLModelData _mLModelData;
        private readonly MLContext _mLContext;
        private ITransformer _mLModel;

        1 reference
        public MLServiceBase(IOptions<MLModelData> mLModelData)
        {
            _mLModelData = mLModelData.Value;
            _mLContext = new MLContext(seed: 0);
            Start();
        }

        1 reference
        private void EvaluateModel()
        {
            var data = _mLContext.Data.LoadFromTextFile<EncryptionCriteria>(
                    path: _mLModelData.ConnectionPath,
                    separatorChar: ',',
                    hasHeader: true);
            var trainTestSplit = _mLContext.Data.TrainTestSplit(data, testFraction: 0.2);
            var trainData = trainTestSplit.TrainSet;
            var testData = trainTestSplit.TestSet;


            var experiment = _mLContext.Auto()
                .CreateMulticlassClassificationExperiment(new MulticlassExperimentSettings
                {
                    MaxExperimentTimeInSeconds = 60,
                    OptimizingMetric = MulticlassClassificationMetric.MacroAccuracy
                });
            var result = experiment.Execute(trainData, labelColumnName: "Algorithm");
            _mLModel = result.BestRun.Model;
            string modelPath = _mLModelData.TrainedModel;
            _mLContext.Model.Save(_mLModel, trainData.Schema, modelPath);
        }
```

Рисунок 3.7 – Фрагмент коду класу MLServiceBase, для навчання та збереження моделі машинного навчання



```
    string modelPath = _mLModelData.TrainedModel;
    _mLContext.Model.Save(_mLModel, trainData.Schema, modelPath);
}

1 reference
protected EncryptionPrediction Predict(EncryptionCriteria encryptionCriteria)
{
    var predictionEngine = _mLContext.Model.CreatePredictionEngine<EncryptionCriteria, EncryptionPrediction>(_mLModel);
    var prediction = predictionEngine.Predict(encryptionCriteria);
    return prediction;
}

1 reference
private void LoadMlModel(string? modelPath)
{
    if (string.IsNullOrEmpty(modelPath))
        throw new ArgumentException();

    try
    {
        using var stream = File.OpenRead(modelPath);
        _mLModel = _mLContext.Model.Load(stream, out _);
    }
    catch (Exception)
    {
        return;
    }
}

1 reference
private void Start()
{
    LoadMlModel(_mLModelData.TrainedModel);

    if (_mLModel == null)
    {
        EvaluateModel();
    }
}
```

Рисунок 3.8 – Фрагмент коду класу MLServiceBase, для завантаження моделі, передбачення результату та ініціалізації сервісу машинного навчання

У конструкторі класу через механізм IOptions отримуються налаштування з конфігурації, що містять шлях до файлу з навчальними даними та шлях для збереження готової моделі. Створюється контекст ML.NET із фіксованим зерном генератора випадкових чисел для відтворюваності результатів. Одразу після ініціалізації викликається метод Start, який намагається завантажити вже навчений файл моделі. Якщо файл відсутній або його завантаження не вдалося, запускається метод EvaluateModel. Метод EvaluateModel завантажує дані для навчання з CSV-файлу, використовуючи тип EncryptionCriteria для коректного парсингу колонок. Потім дані розбиваються на тренувальну та тестову вибірки у пропорції 80%/20%. Застосовується автоматичний експеримент з мультикласової класифікації, де максимальний час пошуку найкращої моделі обмежений 60 секундами, а оптимізаційним критерієм виступає макро-метрика точності (MacroAccuracy). Після завершення експерименту найкраща модель зберігається в пам’яті та серіалізується у файл для подальшого використання. Метод Predict відповідає за

отримання передбачення на основі вхідних критеріїв шифрування. Для цього створюється PredictionEngine, який бере об'єкт EncryptionCriteria, передає його навченій моделі і отримує об'єкт передбачення типу EncryptionPrediction. Результат повертається для подальшої обробки. Завантаження моделі з файлу реалізовано у методі LoadMlModel, який читає файл з диску і завантажує модель у пам'ять. Якщо шлях до моделі некоректний або виникає помилка при читанні, метод просто повертається, залишаючи поле _mLModel незаповненим, що ініціює повторне навчання.

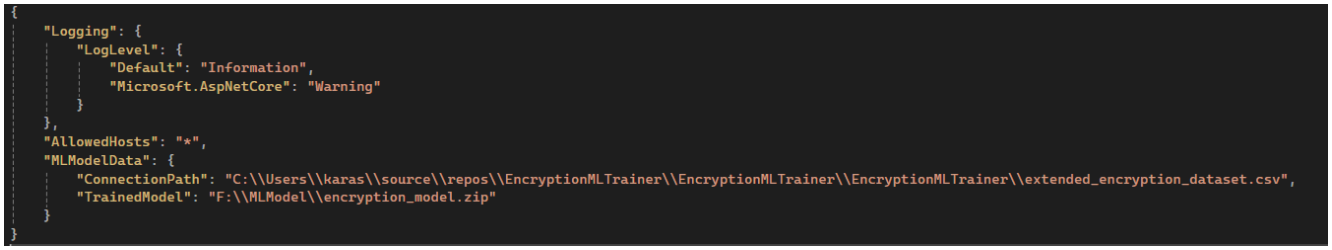
Продовжуючи опис функціоналу серверної частини, зокрема взаємодію з моделлю машинного навчання, варто розглянути структуру, яка відповідає за збереження результатів передбачення. У файлі EncryptionPrediction, який наведено на рис. 3.9 визначено клас, який слугує контейнером для виводу прогнозу, отриманого від ML-моделі.

```
namespace EncryptionScanner.Models
{
    2 references
    public class EncryptionPrediction
    {
        [ColumnName("PredictedLabel")]
        1 reference
        public string Algorithm { get; set; }
    }
}
```

Рисунок 3.9 – Клас для збереження результату передбачення алгоритму шифрування

Він містить одну властивість Algorithm, яка позначена атрибутом [ColumnName("PredictedLabel")]. Цей атрибут вказує на відповідність властивості з іменем колонки у вихідних даних моделі, що дозволяє коректно відобразити передбачене значення.

Далі розглянемо конфігураційний файл appsettings.json, який приводиться на рис. 3.10, що відповідає за налаштування параметрів застосунку, зокрема для логування та роботи з моделлю машинного навчання.



```
{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*",
    "MLModelData": {
        "ConnectionPath": "C:\\Users\\karas\\source\\repos\\EncryptionMLTrainer\\EncryptionMLTrainer\\EncryptionMLTrainer\\extended_encryption_dataset.csv",
        "TrainedModel": "F:\\MLModel\\encryption_model.zip"
    }
}
```


Рисунок 3.10 – Конфігураційний файл appsettings.json

У цьому файлі визначені параметри рівня логування: за замовчуванням встановлено рівень "Information", що означає запис інформаційних повідомлень, а для простору імен Microsoft.AspNetCore встановлено рівень "Warning", щоб фільтрувати менш важливі повідомлення. У розділі MLModelData задаються шляхи до файлу з вхідними даними для навчання моделі (ConnectionPath) та до вже навченого файлу моделі (TrainedModel), які використовуються класами сервісної частини для завантаження даних і моделі машинного навчання. Ці налаштування дозволяють гнучко керувати шляхами до необхідних ресурсів без зміни коду, забезпечуючи зручність при розгортанні та підтримці системи.

Також важливо розглянути початкову конфігурацію та запуск веб-додатку, що виконується у файлі Program.cs, який зображено на рис. 3.11. Цей файл відповідає за налаштування основних сервісів, підключення залежностей, маршрутизацію та запуск застосунку.

```
var builder = WebApplication.CreateBuilder(args);


builder.Services.AddControllersWithViews();

builder.Services.Configure<MLModelData>(builder.Configuration.GetSection("MLModelData"));
builder.Services.AddSingleton<IMLService, MLService>();
var app = builder.Build();


if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");

    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseRouting();

app.UseAuthorization();

app.MapStaticAssets();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}")
    .WithStaticAssets();


app.Run();
```

Рисунок 3.11 – Конфігурація сервісів і маршрутизації у Program.cs

На початку створюється об’єкт builder, який ініціалізує налаштування веб-додатку, отримуючи параметри з аргументів запуску. Далі у контейнер служб додається підтримка контролерів з представленнями (MVC), що дозволяє обробляти HTTP-запити і пов’язувати їх із відповідними контролерами та поданнями. Наступним кроком виконується прив’язка конфігураційного розділу «MLModelData» з файлу appsettings.json до відповідного класу MLModelData, що забезпечує доступ до параметрів моделі машинного навчання через залежності. Потім до контейнера служб додається сервіс MLService, який реалізує інтерфейс IMLService, і реєструється як сінглтон – тобто один екземпляр цього сервісу буде використовуватися упродовж усього часу роботи застосунку. Після завершення конфігурації сервісів, виконується побудова об'єкта app, що представляє сам веб-додаток. За допомогою методів MapStaticAssets та MapControllerRoute встановлюється обробка статичних ресурсів та визначається маршрут за замовчуванням, який вказує, що початковою точкою входу є метод Index контролера Home. Наостанок виконується запуск веб-додатку методом Run, що

ініціює прослуховування HTTP-запитів та взаємодію з користувачем. Файл Program.cs забезпечує базове налаштування і запуск серверної частини, створюючи каркас для подальшої роботи з машинним навчанням та обробки запитів клієнтів.

3.5 Тестування роботи системи

Після завершення розробки та навчання моделі надзвичайно важливо провести її тестування, що є ключовим етапом оцінки працездатності системи в реальних умовах. Тестування дозволяє перевірити, наскільки модель коректно інтерпретує вхідні дані і чи приймає вона правильні рішення у виборі оптимального алгоритму шифрування відповідно до заданих критеріїв. Цей процес передбачає підготовку різноманітних сценаріїв з різними наборами вхідних параметрів, які відображають реальні або можливі умови застосування системи. Подаючи ці вхідні дані на вхід моделі, можна простежити її реакцію і зрозуміти, яким чином вона аналізує кожен параметр, які чинники для неї є ключовими, і як різні характеристики впливають на вибір алгоритму. Завдяки цьому тестуванню можна не лише підтвердити правильність роботи моделі, але й отримати глибше розуміння її логіки, визначити сильні та слабкі сторони, а також можливі напрямки для подальшого вдосконалення. Це дає змогу переконатися, що система здатна адаптуватися до різних вимог безпеки, продуктивності, сумісності з платформою та інших важливих критеріїв, що є надзвичайно важливим для її ефективного використання в реальних умовах.

Обираючи вхідні критерії для системи шифрування, які зображені на рис. 3.12, а саме високий рівень безпеки, високу швидкодію, платформу використання у вигляді веб-додатків і браузерів, відсутність суттєвих обмежень по ресурсах, роботу з дуже великим обсягом даних (понад 10 ГБ), необхідність відповідності стандартам FIPS/ISO, підтримку масштабованості та паралельності, наявність автентифікації та високу стійкість до помилок, система прийшла до висновку, що найоптимальнішим вибором є алгоритм AES.

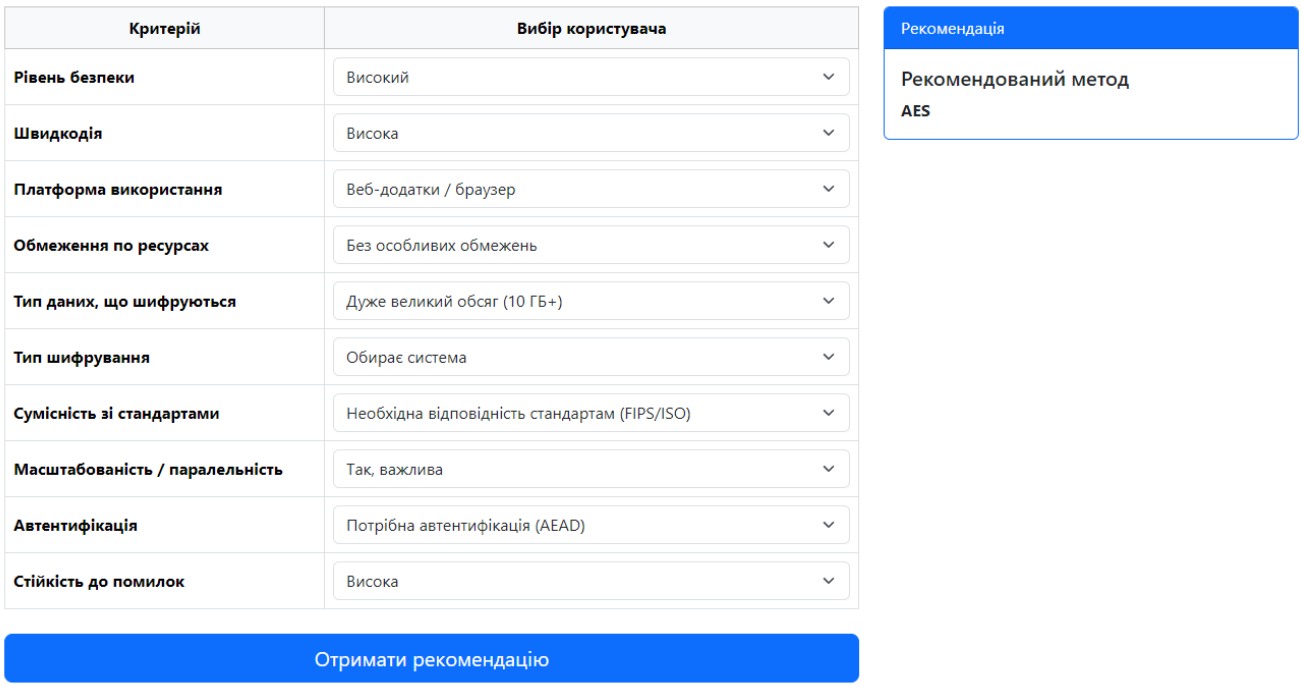


Рисунок 3.12 – Вхідні параметри системи для вибору алгоритму шифрування AES

Рівень безпеки AES є одним із найвищих серед сучасних симетричних алгоритмів шифрування, що робить його надійним вибором для захисту конфіденційних даних у критично важливих системах. Висока швидкодія AES забезпечується ефективною апаратною підтримкою на більшості сучасних процесорів, що дозволяє швидко обробляти великі обсяги інформації без значних затримок. Враховуючи платформу використання у вигляді веб-додатків і браузерів, AES має широке розповсюдження і підтримується більшістю сучасних веб-браузерів, що забезпечує сумісність і простоту інтеграції. Відсутність значних обмежень по ресурсах дозволяє застосовувати AES у його повній потужності, що особливо важливо при роботі з великими обсягами даних. Щодо обробки дуже великих обсягів (понад 10 ГБ), AES підтримує режим роботи з паралельністю, що підвищує продуктивність і забезпечує масштабованість системи. Вимога відповідності стандартам FIPS/ISO є одним із ключових чинників вибору AES, оскільки цей алгоритм є офіційно сертифікованим і рекомендованим для використання в державних та комерційних інформаційних системах. Підтримка автентифікації в AES, особливо у режимі GCM, гарантує цілісність і достовірність

даних, що є критично важливим для безпечного обміну інформацією. Нарешті, висока стійкість до помилок забезпечує безперервність і надійність роботи системи навіть у разі виникнення незначних збоїв або пошкоджень даних. Тому з огляду на всі ці критерії, алгоритм AES є найбільш збалансованим і підходящим рішенням для даного набору вимог.

Обираючи для системи такі критерії, що зображені на рис. 3.13, а саме – низький рівень безпеки, середня швидкодія, використання на веб-додатках чи в браузері, обмеження по ресурсах у вигляді мінімального енергоспоживання, великий обсяг даних від 500 МБ до 10 ГБ, можливість використовувати експериментальні алгоритми, не критичну масштабованість і паралельність, відсутність обов'язкової автентифікації, а також середню стійкість до помилок, система прийшла до висновку, що оптимальним вибором є алгоритм ChaCha20.

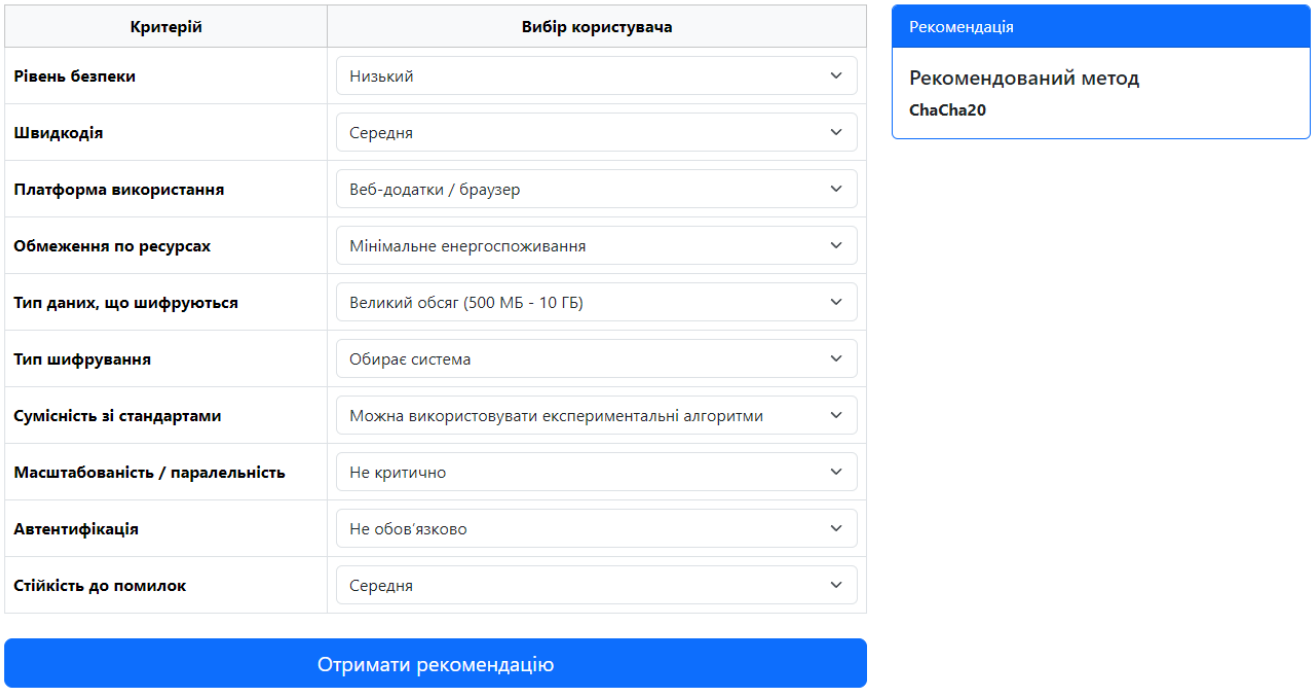


Рисунок 3.13 – Вхідні параметри системи для вибору алгоритму шифрування ChaCha20

Цей алгоритм є високопродуктивним і водночас економним з точки зору енергоспоживання, що робить його ідеальним для застосування в середовищах з

обмеженими ресурсами, таких як мобільні веб-додатки або браузери, де важлива ефективність без зайвого навантаження на батарею. ChaCha20 забезпечує достатній рівень безпеки для задач, де не потрібно суворого відповідності жорстким стандартам, оскільки він підтримує використання експериментальних та сучасних криптографічних підходів, що дозволяє впроваджувати інновації. Оскільки масштабованість і паралельність не є критичними, відсутність високої складності алгоритму позитивно впливає на швидкодію, забезпечуючи середню продуктивність, що відповідає вимогам системи. Відсутність обов'язкової автентифікації зменшує потребу у складних механізмах контролю доступу, а середня стійкість до помилок узгоджується з загальною легкістю та простотою ChaCha20, що робить його оптимальним вибором для таких умов.

Враховуючи наступні вхідні критерії, що зображені на рис. 3.14, а саме високий рівень безпеки, необов'язкову високу швидкодію, використання на серверних системах або в хмарних середовищах, відсутність обмежень по ресурсах, малий обсяг даних до 10 МБ, необхідність застосування асиметричного шифрування, сувору відповідність стандартам FIPS/ISO, незначну роль масштабованості та паралельності, потребу в автентифікації та середню стійкість до помилок, система зробила вибір на користь алгоритму RSA.

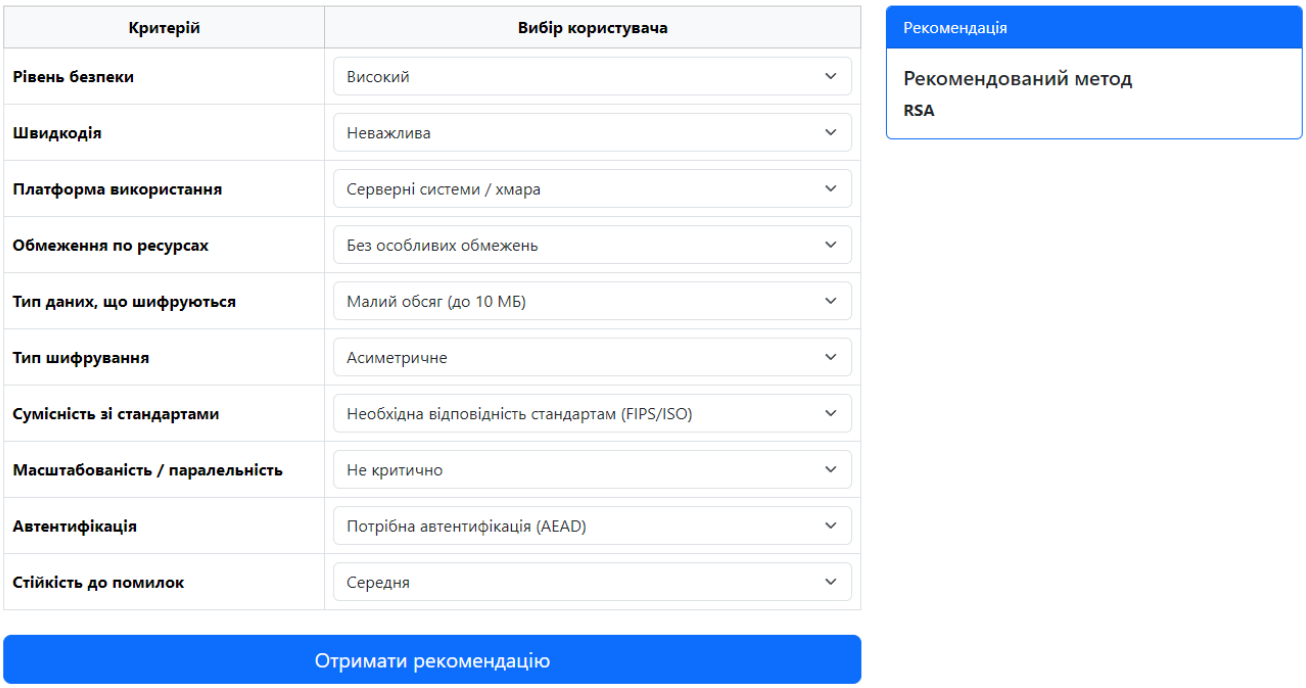


Рисунок 3.14 – Вхідні параметри системи для вибору алгоритму шифрування RSA

Цей криптографічний алгоритм ідеально підходить для серверних і хмарних застосувань, де важливо забезпечити високий рівень захисту даних і надійну автентифікацію користувачів або пристроїв. RSA належить до асиметричних алгоритмів, що дає змогу безпечно обмінюватися ключами та підтверджувати особистість за допомогою цифрових підписів, що є критично важливим у середовищах із суворими вимогами до безпеки. Незважаючи на те, що швидкодія RSA нижча порівняно з симетричними алгоритмами, це не є проблемою для систем з малим обсягом даних і необмеженими ресурсами, де пріоритетом є саме безпека, а не продуктивність. Відповідність міжнародним стандартам FIPS та ISO гарантує, що використання RSA відповідає найвищим вимогам галузі, що особливо важливо у корпоративних і державних середовищах. Середня стійкість до помилок не є критичною, оскільки серверні системи зазвичай забезпечують додаткові механізми контролю якості передачі даних. Відсутність потреби у масштабованості і паралельності не впливає на вибір, адже RSA найкраще підходить саме для сценаріїв з обмеженим обсягом даних, де основним є надійність і безпека.

Враховуючи всі задані параметри, RSA є оптимальним рішенням для реалізації системи шифрування в цьому конкретному випадку.

У випадку, коли задані такі вхідні критерії, що зображені на рис. 3.15, а саме – високий рівень безпеки, неважлива швидкодія, використання у вбудованих системах або пристроях Інтернет речей (IoT), наявність обмежень по обчислювальних ресурсах, малий обсяг даних (до 10 МБ), потреба в асиметричному типі шифрування, обов'язкова відповідність міжнародним стандартам FIPS/ISO, незначна роль масштабованості й паралельності, необхідність автентифікації, а також висока стійкість до помилок, система обрала алгоритм ECC.

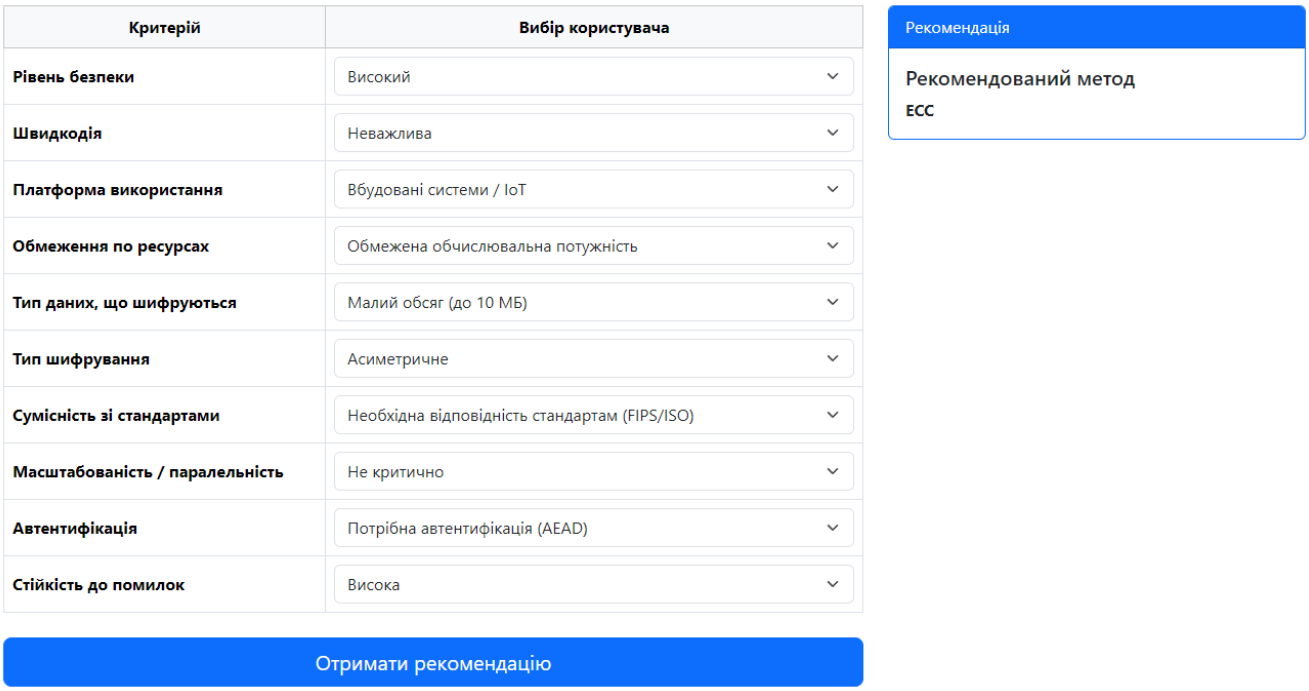


Рисунок 3.15 – Вхідні параметри системи для вибору алгоритму шифрування ECC

Цей вибір є цілком виправданим і оптимальним, оскільки ECC забезпечує такий самий рівень криптографічного захисту, як і класичні асиметричні алгоритми на кшталт RSA, але при цьому потребує значно менше обчислювальних ресурсів. Це робить ECC ідеальним кандидатом для реалізації у вбудованих системах, де розмір пам'яті, швидкодія процесора або енергоспоживання є обмеженими. Малий розмір ключів, характерний для ECC, дає змогу знизити навантаження на систему

при збереженні високого рівня безпеки, що особливо актуально в середовищах IoT. Підтримка ECC у міжнародних криптографічних стандартах, таких як FIPS і ISO, забезпечує відповідність регуляторним вимогам і можливість використання в промислових або державних застосуваннях. Необхідність автентифікації реалізується за допомогою цифрових підписів на основі еліптичних кривих, що є ефективним і легким для валідації навіть у слабких пристроях. Крім того, ECC має високу стійкість до помилок, що критично важливо в умовах нестабільного бездротового зв'язку або нестандартного середовища функціонування, характерного для IoT. Тому, на основі зазначених параметрів, ECC виявляється найкращим варіантом, що поєднує в собі компактність, безпеку, відповідність стандартам та ефективність у середовищах з обмеженими ресурсами.

У підсумку проведеного тестування системи було підтверджено її здатність ефективно аналізувати вхідні критерії та обирати оптимальний криптографічний алгоритм залежно від контексту використання, вимог до безпеки, продуктивності та обмежень ресурсів. У кожному зі сценаріїв система враховувала не лише загальні параметри, як-от обсяг даних або тип платформи, а й більш специфічні фактори, такі як необхідність автентифікації, підтримка стандартів чи стійкість до помилок. В результаті для кожного унікального набору характеристик був запропонований найбільш доцільний алгоритм – AES, ChaCha20, RSA або ECC – із ґрунтовним обґрунтуванням вибору. Це свідчить про коректну роботу машинної моделі, її відповідність поставленим цілям та здатність забезпечувати адаптивність до широкого спектра практичних умов. Таким чином, система довела свою ефективність як інтелектуальний інструмент для підтримки рішень у сфері інформаційної безпеки.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було проведено комплексне дослідження сучасних методів шифрування та розроблено автоматизовану систему для їх вибору з використанням технологій машинного навчання. Стан питання захисту даних свідчить про зростаючу потребу в ефективних криптографічних рішеннях, оскільки традиційні підходи до вибору алгоритмів шифрування часто не враховують динамічних умов експлуатації, що знижує їх ефективність у реальних інформаційних системах. Проведений аналіз сучасних методів шифрування, зокрема симетричних (AES, ChaCha20), асиметричних (RSA, ECC) та гібридних, дозволив визначити ключові критерії оцінки їх ефективності. Ці критерії стали основою для розробки системи, яка забезпечує автоматизований вибір оптимального алгоритму шифрування.

Для реалізації системи було обґрунтовано вибір клієнт-серверної архітектури, яка забезпечує гнучкість, масштабованість та зручність використання. Як технологічний стек використано мову програмування C# для серверної частини та бібліотеку ML.NET для інтеграції моделі машинного навчання, що дозволило автоматизувати процес вибору алгоритму на основі вхідних критеріїв, таких як обсяг даних, вимоги до продуктивності та рівень безпеки. Розроблено зручний користувацький інтерфейс, який забезпечує інтуїтивну взаємодію з системою, та серверну частину, яка обробляє запити й реалізує логіку машинного навчання. Проведене тестування підтвердило працездатність системи, її здатність точно оцінювати ефективність алгоритмів шифрування та обирати оптимальний метод у реальних сценаріях використання.

Теоретичним результатом дослідження стало обґрунтування підходу до автоматизованого вибору методів шифрування на основі машинного навчання, який забезпечує адаптивність до динамічних умов експлуатації, що є новим у порівнянні з традиційними статичними методами. Практичним результатом є розроблена система, яка може бути застосована в інформаційних системах для підвищення ефективності захисту даних у таких галузях, як банківська справа, електронна комерція та державне управління. Достовірність результатів

підтверджується використанням апробованих стандартів шифрування, сучасних технологій машинного навчання та ретельним тестуванням системи.

На основі отриманих результатів сформульовано практичні рекомендації: для забезпечення конфіденційності даних у веб-додатках доцільно використовувати розроблену систему для автоматичного вибору алгоритмів шифрування, що дозволяє оптимізувати баланс між безпекою та продуктивністю. Рекомендується інтегрувати систему в інформаційні платформи, де захист даних є критично важливим, а також проводити регулярне оновлення моделей машинного навчання з урахуванням нових кіберзагроз і алгоритмів шифрування. Подальші дослідження можуть бути спрямовані на розширення функціональності системи шляхом додавання підтримки нових алгоритмів і вдосконалення моделей машинного навчання для підвищення точності вибору.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Роль шифрування у захисті даних. URL: https://surli.cc/zeaqge (дата звернення: 02.11.2024)

2. Горбань І. І. Інформаційна безпека: основи криптографії. Київ: НТУУ "КПІ", 2016. 320 с.

3. Симетричне шифрування. URL: https://surli.cc/tcjize (дата звернення: 08.11.2024)

4. Що таке AES шифрування. URL: https://memory.net.ua/info/aes-shifruvannja?srsltid=AfmBOooM5nkxZo2t0zSBGCri4i91xdfC4coPjZFlaKa1Xxj0qafGLDLi (дата звернення: 19.11.2024)

5. Що таке ChaCha20. URL: https://surl.li/dsznfq (дата звернення: 19.11.2024)

6. Stallings W. Cryptography and Network Security. Boston: Pearson, 2020. 768 с.

7. Асиметричні алгоритми шифрування. URL: https://surl.lu/qiqvhb (дата звернення: 26.11.2024)

8. Ткачук М. В. Криптографічні методи захисту інформації. Львів: Видавництво Львівської політехніки, 2018. 280 с.

9. Алгоритм шифрування RSA. Види атак на нього. URL: https://dou.ua/forums/topic/43026/ (дата звернення: 02.12.2024)

10. Що таке ECC і навіщо його використовувати. URL: https://surl.li/lghtfs (дата звернення: 07.12.2024)

11. Визначення гібридного шифрування. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/hybrid-encryption (дата звернення: 08.12.2024)

12. Ефективність алгоритму. URL: https://surl.lu/guqcdl (дата звернення: 15.12.2024)

13. Розуміння Клієнт-Серверної Архітектури на прикладах. URL: https://surl.li/ugkxmo (дата звернення: 20.12.2024)

14. Що таке Bootstrap і навіщо він потрібен. URL: https://itmaster.biz.ua/programming/web-prohramuvannia/bootstrap.html (дата звернення: 25.12.2024)

15. ASP.NET MVC Framework. URL: https://ru.wikipedia.org/wiki/ASP.NET_MVC_Framework (дата звернення 07.01.2025)

16. MVC: шаблон проектування архітектури додатку. URL: https://surli.cc/lofrys (дата звернення: 15.01.2025)

17. Що таке машинне навчання: як працює та де використовується. URL: https://gigacloud.ua/articles/shho-take-mashynne-navchannya-yak-praczyuye-ta-de-vykorystovuyetsya/ (дата звернення: 18.01.2025)

18. Перші кроки з ML.NET: як навчити машину розпізнавати об’єкти. URL: https://dou.ua/forums/topic/34961/ (дата звернення: 23.01.2025)

19. ML.NET. URL: https://ru.wikipedia.org/wiki/ML.NET (дата звернення: 02.02.2025)

20. Роль ШІ та машинного навчання в кібербезпеці у 2025 році. URL: https://lazarusalliance.com/uk/the-role-of-ai-and-machine-learning-in-cybersecurity-in-2025/ (дата звернення: 06.02.2025)

21. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge: MIT Press, 2016. 800 с.

22. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Sebastopol: O’Reilly Media, 2022. 856 с.

23. Machine Learning, ML. URL: https://www.it.ua/knowledge-base/technology-innovation/machine-learning (дата звернення: 07.02.2025)

24. Що таке класифікаційне машинне навчання. URL: https://itwiki.dev/data-science/ml-reference/ml-glossary/multi-class-classification (дата звернення: 18.02.2025)

25. Лінійні регресії. URL: https://w3schoolsua.github.io/ai/ai_regressions.html#gsc.tab=0 (дата звернення: 09.02.2025)

26. Огляд алгоритмів машинного навчання. URL: https://www.zfort.com.ua/blog/cekretni-sili-mashinnogo-navchannya-oglyad-algoritmiv-mashinnogo-navchannya (дата звернення: 27.02.2025)

27. Troelsen A. Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming. New York: Apress, 2021. 1372 c.

28. Brown T. B. Machine Learning for Cybersecurity. New York: Apress, 2023. 420 c.

29. Аналіз алгоритмів машинного навчання в ML.NET. https://crust.ust.edu.ua/items/99ef6416-4214-4f33-bc88-da4cb3aad2c6 (дата звернення: 06.03.2025)

30. Завантаження даних для навчання у будівельник моделей. URL: https://learn.microsoft.com/ru-ru/dotnet/machine-learning/how-to-guides/load-data-model-builder (дата звернення: 10.03.2025)