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

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему

СИСТЕМА ІНТЕГРАЦІЇ КРИПТОГРАФІЧНИХ МЕТОДІВ У СЕРВІСИ ДЛЯ ЗАХИСТУ ДАНИХ КОРИСТУВАЧІВ

Виконав: студент групи 2КІ-23
Спеціальності 123 «Комп'ютерна інженерія»
Гапченко В.С.
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 СТУДЕНТУ

Гапченко Володимир Сергійович
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи Система інтеграції криптографічних методів у сервіси для захисту даних користувачів
Науковий керівник роботи к.н.т доцент Захарова Марія В'ячеславівна
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від " 07 " жовтня 2024 року № 68У.
2. Строк подання студентом випускної роботи 10.06.2025
3. Вихідні дані до випускної роботи Система інтеграції криптографічних методів у сервіси для захисту даних користувачів включає аналіз сучасних підходів до оцінки безпеки хмарних платформ, виявлення їхніх вразливостей та потенційних загроз, а також вивчення існуючих стандартів і нормативно-правових вимог щодо захисту інформації. В рамках роботи передбачено розробку концептуальної моделі інтеграції криптографічних засобів у хмарні сервіси, її апробацію в експериментальному середовищі та формулювання рекомендацій для підвищення рівня безпеки користувацьких даних у хмарних обчисленнях.
4. Зміст випускної роботи (перелік питань, які потрібно розробити) визначення актуальності теми, мети, завдання, об'єкту, предмету, детальний огляд та порівняння аналогів системи біометричної аутентифікації, враховуючи їхні переваги, недоліки та сучасні тенденції розвитку
5. Дата видачі завдання 15.09.2024р.

.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання з підписами наукового керівника і студента
1	Вступ	13.10.2024	
2	Розділ 1. Теоретичні засади криптографічного захисту даних	17.12.2024	
3	Розділ 2. Методи інтеграції криптографії у сервіси	07.03.2025	
4	Розділ 3. Розробка та тестування системи інтеграції криптографічних методів	09.04.2025	
5	Висновки	01.06.2025	
6	Оформлення випускної роботи (чистовий варіант)	03.06.2025	
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 _____ Гапченко В.С
(підпис) (прізвище та ініціали)

Керівник роботи _____ Захарова М.В.
(підпис) (прізвище та ініціали)

Анотація

У кваліфікаційній роботі бакалавра запропоновано браузерну систему захисту даних, що реалізує шифрування, дешифрування та перевірку цілісності повідомлень за допомогою сучасних криптографічних методів — AES-GCM, PBKDF2 та HMAC. Основна мета розробки полягає у створенні легкого клієнтського рішення, яке не потребує серверної інфраструктури та забезпечує безпечну обробку інформації локально у веббраузері.

У роботі проаналізовано особливості реалізації криптографічних алгоритмів у середовищі JavaScript із використанням Web Crypto API. Побудовано функціональний вебінтерфейс, який дозволяє здійснювати повний цикл обробки повідомлень: введення, шифрування, генерацію MAC-коду, дешифрування та обробку помилок.

Проведено тестування працездатності системи та оцінено ефективність рішень для задач локального захисту даних. Результати підтверджують придатність прототипу для впровадження в освітні, комерційні та внутрішні сервіси, що не використовують зовнішні сховища.

Кваліфікаційна робота бакалавра містить 72 аркуші пояснювальної записки, 15 рисунків, 17 таблиць і 1 додатки.

Ключові слова: криптографія, AES-GCM, HMAC, Web Crypto API, захист інформації, вебінтерфейс, шифрування, JavaScript.

Annotation

The bachelor's qualification work proposes a browser-based data protection system that implements encryption, decryption, and message integrity verification using modern cryptographic methods — AES-GCM, PBKDF2, and HMAC. The main goal of the development is to create a lightweight client-side solution that does not require server infrastructure and ensures secure local data processing in the web browser.

The paper analyzes the implementation features of cryptographic algorithms in JavaScript using the Web Crypto API. A functional web interface has been developed to support the full message processing cycle: input, encryption, MAC generation, decryption, and error handling.

System performance testing was conducted, and the effectiveness of the solution for local data protection tasks was evaluated. The results confirm the suitability of the prototype for implementation in educational, commercial, and internal services that do not rely on external storage.

The bachelor's qualification work contains 72 pages of explanatory note, 15 figures, 17 tables, and 1 appendices.

Keywords: cryptography, AES-GCM, HMAC, Web Crypto API, data protection, web interface, encryption, JavaScript.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

Позначення	Розшифрування
AES	Advanced Encryption Standard
HMAC	Hashed Message Authentication Code
PBKDF2	Password-Based Key Derivation Function 2
GCM	Galois/Counter Mode
SHA-256	Secure Hash Algorithm 256
Web Crypto API	Web Cryptography API
IV	Initialization Vector
MAC	Message Authentication Code
HTML	HyperText Markup Language
JS	JavaScript
UI	User Interface
DOM	Document Object Model
JSON	JavaScript Object Notation
UTF-8	Unicode Transformation Format – 8-bit
Blob	Binary Large Object

ВСТУП

Актуальність теми. У сучасному цифровому середовищі безпека даних користувачів набуває критичного значення. В умовах стрімкого розвитку інформаційних технологій, зокрема веб- і хмарних сервісів, зростає кількість кіберзагроз, пов’язаних із несанкціонованим доступом, крадіжкою персональної інформації та підробкою даних. Ефективне впровадження криптографічних методів у сервіси забезпечує конфіденційність, цілісність та автентичність даних, що, у свою чергу, є необхідною умовою для побудови довіри користувачів до цифрових платформ. Актуальність теми зумовлена потребою в розробці та впровадженні надійної, продуктивної та масштабованої системи інтеграції криптографічних алгоритмів у сервіси для захисту користувацьких даних.

Метою кваліфікаційної роботи є розробка системи інтеграції криптографічних методів у сервіси для підвищення рівня захисту даних користувачів, з урахуванням особливостей сучасних технологій та вимог до продуктивності.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

- Проаналізувати сучасні підходи до захисту даних у вебсередовищі.
- Визначити оптимальні криптографічні методи для інтеграції у браузерні сервіси.
- Розробити та реалізувати прототип системи з функціями шифрування, дешифрування та HMAC.
- Провести тестування системи на коректність, швидкодію та стійкість до помилок.
- Оцінити ефективність та придатність розробленого рішення для подальшого впровадження.

Об’єктом дослідження є інформаційні сервіси, що потребують захисту даних користувачів.

Предметом дослідження є процеси інтеграції криптографічних методів у ці сервіси для забезпечення конфіденційності, цілісності й доступності інформації.

У роботі використано такі **методи**:

– Криптографічне шифрування (AES-GCM) — для забезпечення конфіденційності повідомлень у браузері.

– Деривація ключа (PBKDF2 + SHA-256) — для генерації надійного симетричного ключа з введеного пароля.

– HMAC (SHA-256) — для перевірки цілісності та автентичності даних.

– Експериментальне моделювання — для перевірки роботи системи в реальному середовищі.

– Функціональне тестування — для оцінки сумісності в різних браузерах.

– Аналіз помилок — для обробки ситуацій з некоректним ключем чи пошкодженим шифртекстом.

Інформаційну основу дослідження склали міжнародні стандарти безпеки (ISO/IEC 27001, ISO/IEC 18033), наукові статті, підручники з криптографії, документація до криптографічних бібліотек (OpenSSL, BouncyCastle), аналітичні звіти з безпеки провідних IT-компаній (Microsoft, Google, Cisco), а також результати практичного тестування авторської системи.

Практичне значення роботи полягає в аналітичному обґрунтуванні ефективності застосування криптографічних методів для захисту даних у хмарних, веб- та мобільних сервісах. Представлені моделі та узагальнення можуть бути використані як методологічна база для подальшої розробки захищених інформаційних систем, адаптованих до потреб малого та середнього бізнесу, державних структур та освітніх платформ.

Апробація результатів. Основні результати дослідження були апробовані під час участі у XVII Студентській науково-практичній конференції «Тенденції розвитку IT-технологій в Україні», яка відбулася 26 березня 2025 року на базі Черкаського державного бізнес-коледжу. У межах напряму «Цифрова безпека: проблеми та перспективи» було представлено тези доповіді на тему «Система інтеграції криптографічних методів у сервіси для захисту даних користувачів»,

які отримали позитивні відгуки від наукового керівництва та учасників конференції.

Структура і обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел (50 найменувань) та додатків. Загальний обсяг роботи становить 72 сторінок комп’ютерного тексту.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ЗАСАДИ КРИПТОГРАФІЧНОГО ЗАХИСТУ ДАНИХ

1.1 Основні принципи криптографії

Криптографія — це наука про методи забезпечення конфіденційності, цілісності, автентичності та незаперечності інформації шляхом її перетворення у форму, незрозумілу для сторонніх осіб. Вона є однією з ключових галузей інформаційної безпеки, що формує фундамент захисту даних в умовах цифрового середовища.

Криптографія базується на використанні математичних алгоритмів, які реалізують процеси шифрування (перетворення відкритого тексту в зашифрований) та дешифрування (зворотне перетворення). Залежно від того, який тип ключа використовується — симетричний чи асиметричний — криптографія поділяється на відповідні підкласи [7].

Історія криптографії налічує тисячоліття. Перші згадки про шифрування відомі ще з часів Давнього Єгипту, де ієрогліфи змінювалися для приховування змісту написів, що вважається одними з перших проявів криптографії. У VI–IV століттях до нашої ери в Давній Греції використовували скіталу — спеціальний пристрій для шифрування, що базувався на зміщенні букв. У I столітті до нашої ери з’явилося так зване Цезареве шифрування — один із перших формалізованих шифрів, у якому кожна літера замінювалася на третю наступну в алфавіті.

У період IX–XII століть в арабському світі розвивалися основи криптоаналізу, зокрема було закладено поняття частотного аналізу. У XVI столітті з’явився шифр Віженера — поліалфавітний метод шифрування, який тривалий час вважався нерозгаданим. У XX столітті криптографія стала невіддільною частиною військової доктрини. Одним із найвідоміших прикладів є шифрувальна машина Enigma, що використовувалась під час Другої світової війни.

Починаючи з 1970-х років, розпочався етап розвитку комп’ютерної криптографії. Було створено низку ключових алгоритмів, таких як DES, RSA,

AES і ECC. Саме в цей період закладалися сучасні стандарти та відбувався перехід до цифрових систем захисту інформації. У сучасну епоху криптографія розвивається в напрямі постквантових алгоритмів, активно застосовується в блокчейні, цифрових підписах, хмарних технологіях та навіть у повсякденному житті користувачів.

У сучасних умовах цифрової трансформації, коли обсяг оброблюваної інформації стрімко зростає, питання забезпечення її безпеки набуває особливої ваги. Криптографія, як ключовий інструмент у структурі інформаційної безпеки, реалізує низку фундаментальних принципів, на яких ґрунтується захист даних від несанкціонованого доступу, викривлення або зловживання. До основних принципів криптографічного захисту традиційно відносять конфіденційність, цілісність, доступність, автентифікацію та незаперечуваність. Розглянемо зміст кожного з них крізь призму прикладного криптозахисту [3].

Конфіденційність передбачає забезпечення недоступності інформації для сторонніх осіб, які не мають відповідного дозволу на її перегляд чи використання. У криптографії цей принцип реалізується переважно за допомогою алгоритмів симетричного та асиметричного шифрування. Шифрування перетворює відкриті дані у зашифровану форму, зміст якої зрозумілий лише у разі наявності відповідного ключа. Так, у разі застосування алгоритму AES (Advanced Encryption Standard) або RSA (Rivest–Shamir–Adleman), дані захищені навіть у разі їх перехоплення.

Цілісність гарантує, що дані не були змінені, знищені чи викривлені у процесі зберігання або передавання. Одним із основних інструментів реалізації цього принципу є хеш-функції (наприклад, SHA-2 або SHA-3), які генерують унікальне хеш-значення на основі вхідних даних. У разі несанкціонованої модифікації інформації хеш значно змінюється, що дозволяє виявити порушення цілісності. Цей механізм є критично важливим для захисту баз даних, електронної документації та мережевих протоколів [4].

Принцип доступності означає, що дані повинні бути доступні для авторизованих користувачів у момент потреби. У контексті криптографічного

захисту доступність забезпечується через оптимізацію обчислювальних процесів, належне управління криптографічною інфраструктурою, а також захист від атак типу відмова в обслуговуванні (DoS). Важливо, щоб реалізація шифрування не створювала додаткового навантаження на системи і не обмежувала правомірний доступ до інформаційних ресурсів [45].

Автентифікація визначає механізми перевірки автентичності користувача або джерела інформації (див. табл. 1.1). Вона забезпечується за допомогою цифрових сертифікатів, токенів, криптографічних ключів та протоколів (наприклад, TLS, OAuth). Застосування публічних ключів у інфраструктурі відкритих ключів (PKI) дозволяє реалізувати надійне підтвердження особи, а багатофакторна автентифікація (2FA, MFA) додатково посилює рівень захисту, унеможливлюючи несанкціонований доступ навіть у разі компрометації одного з елементів аутентифікації (див табл 1.1).

Таблиця 1.1 – Порівняльна характеристика основних принципів криптографічного захисту даних

№	Принцип	Сутність	Криптографічні засоби реалізації	Приклади застосування
1	**2**	**3**	**4**	**5**
1	Конфіденційність	Обмеження доступу до інформації лише авторизованим суб'єктам	Алгоритми шифрування (AES, RSA, ECC)	Зашифроване зберігання даних, захищена передача повідомлень
2	Цілісність	Забезпечення незмінності інформації під час її обробки або передавання	Хеш-функції (SHA-2, SHA-3), контрольні суми	Захист електронних документів, перевірка цілісності файлів
3	Доступність	Гарантія отримання доступу до інформації авторизованими користувачами у потрібний момент	Устатковані протоколи, стійкість до DoS-атак	Інтернет-банкінг, сервіси електронної пошти
4	Автентифікація	Перевірка достовірності особи або джерела інформації	Сертифікати, токени, PKI, TLS, HMAC	Аутентифікація при вході в систему, перевірка підпису документа
5	Незаперечуваність	Неможливість відмови суб'єкта від факту здійснення дії або передавання інформації	Цифровий підпис, протокол логування	Електронний документообіг, підписання електронних угод

Джерело: створено автором на основі даних [22]

Незаперечуваність — це гарантія того, що учасник інформаційного обміну не зможе заперечити факт здійснення певної дії або передавання даних. Цей принцип реалізується, зокрема, через використання цифрових підписів. Підпис, сформований за допомогою закритого ключа, може бути верифікований лише з використанням відповідного відкритого ключа, що забезпечує юридичну силу транзакцій, електронних документів і повідомлень у правових і комерційних взаєминах.

Принципи, викладені у таблиці 1.1, є базовими у побудові сучасних систем захисту інформації. Вони визначають функціональні вимоги до криптографічних протоколів і є основою при проектуванні безпечного програмного забезпечення, вебсервісів та цифрової інфраструктури загалом. Їх повноцінна реалізація можлива лише на основі глибокого математичного підґрунтя, що буде розглянуто у наступному підрозділі [8].

Типологія криптографічних методів визначається насамперед способом використання ключів при шифруванні та розшифруванні даних. Сучасна криптографія передбачає три основні типи: симетричну, асиметричну та гібридну. Кожен з них має свої особливості, переваги та обмеження, що визначають сферу його застосування.

Симетричні криптосистеми передбачають використання одного і того ж ключа як для шифрування, так і для дешифрування інформації. Такий підхід забезпечує високу швидкодію та простоту реалізації, однак вимагає безпечного способу обміну ключем, що становить одну з головних вразливостей.

До прикладів симетричних алгоритмів належать AES (Advanced Encryption Standard), який є сучасним стандартом блокового шифрування з довжиною блоку 128 біт і ключами на 128, 192 або 256 біт. Іншим прикладом є DES (Data Encryption Standard) — історично важливий алгоритм, який сьогодні вважається застарілим через недостатню довжину ключа. Також варто згадати ChaCha20 — сучасний потоковий шифр, що набув популярності у мобільних застосунках і web-середовищах завдяки своїй швидкодії та надійності.

Симетрична криптографія широко застосовується у VPN-з’єднаннях, шифруванні файлів і жорстких дисків, а також у вбудованих пристроях, зокрема в Інтернеті речей (IoT).

Асиметричні криптосистеми функціонують за принципом використання пари ключів — відкритого (публічного) та закритого (приватного). Особливість полягає в тому, що дані, зашифровані одним ключем, можуть бути розшифровані лише іншим, що дозволяє забезпечити захист інформації без необхідності попереднього обміну ключами. Асиметричний підхід також широко використовується для реалізації цифрового підпису та автентифікації.

До основних алгоритмів асиметричної криптографії належать RSA — найбільш поширений метод, заснований на складності факторизації великих чисел. ElGamal — алгоритм, побудований на математичній задачі дискретного логарифмування. Ще одним сучасним методом є криптографія на еліптичних кривих (ECC), яка забезпечує високий рівень безпеки навіть за короткої довжини ключів.

Асиметричні алгоритми використовуються у сфері захищеної передачі ключів у протоколах TLS/SSL, в електронних сертифікатах та інфраструктурі відкритих ключів (PKI), а також у цифрових підписах.

Гібридні криптографічні системи поєднують переваги симетричних та асиметричних методів. Зазвичай симетричне шифрування застосовується безпосередньо до передаваних даних, тоді як асиметричне — для безпечної передачі симетричного ключа. Такий підхід дозволяє одночасно досягти високої швидкості обробки інформації й надійного захисту ключів. Типовим прикладом є протокол TLS (Transport Layer Security), у якому спочатку здійснюється обмін ключами за допомогою RSA або ECDHE, після чого основний трафік шифрується за допомогою алгоритмів AES або ChaCha20.

Гібридна криптографія стала де-факто стандартом у більшості сучасних комунікаційних протоколів, веб-застосунків та мобільних додатків (див. рис.1.1).

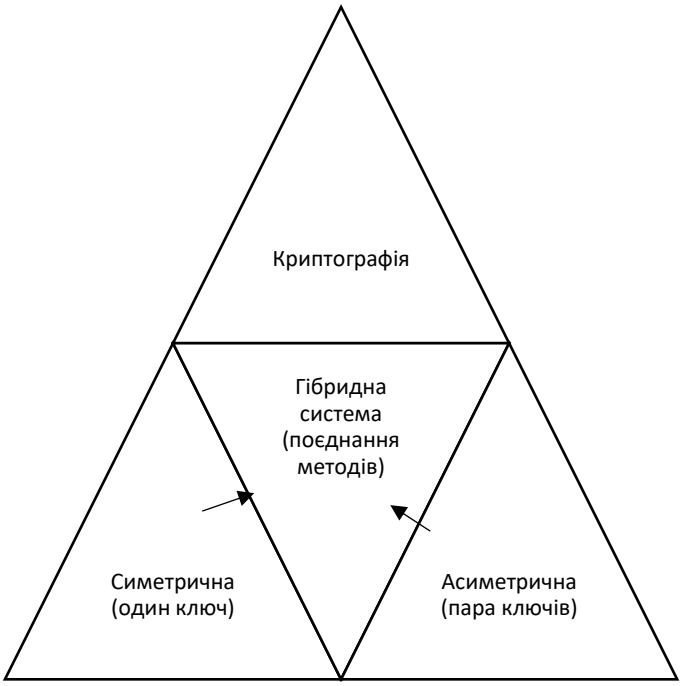

Рисунок 1.1 – Порівняльна блок-схема типів криптографії

Розуміння типів криптографії є ключовим для правильного вибору алгоритмів у конкретних умовах використання — від локального шифрування файлів до захисту міжмережевих протоколів. У наступних підрозділах буде розглянуто математичну базу, на якій ґрунтуються ці методи, та їх реалізацію у практичних алгоритмах.

1.2 Математичні основи криптографії

Теорія чисел є фундаментальною основою для сучасної криптографії, оскільки більшість алгоритмів базуються на властивостях простих чисел, складності факторизації великих чисел та операціях у модульній арифметиці. Розуміння базових понять теорії чисел дозволяє пояснити принципи роботи алгоритмів типу RSA, ElGamal, а також хеш-функцій та цифрових підписів [10].

Просте число — це натуральне число більше за 1, яке має лише два дільники: 1 та саме себе. У криптографії прості числа використовуються як основа для створення великих чисел, які є добутком двох простих множників, — саме така структура забезпечує складність зворотного обчислення (факторизації).

Це створює основу для односторонніх функцій — функцій, які легко обчислюються, але важко інвертуються без додаткової інформації (секретного ключа).

Обчислення найбільшого спільного дільника (НСД) є важливою операцією в криптографії. Воно знаходить широке застосування при генерації відкритих і закритих ключів у криптосистемі RSA, де необхідно визначати взаємно прості числа. Також обчислення НСД використовується для знаходження обернених елементів у модульній арифметиці, що є ключовим для алгоритмів шифрування, цифрового підпису та автентифікації. Крім того, ця операція відіграє важливу роль в аналізі чисел на взаємну простоту, що є фундаментальним для багатьох криптографічних процедур.

Алгоритм Евкліда — один з найефективніших способів обчислення НСД двох цілих чисел a та b, де $a>b$.

$$\gcd(a,b) = \begin{cases} a, \text{якщо } b = 0 \\ \gcd(b, a \bmod b), \text{ інакше} \end{cases} \quad (1.1)$$

Розширена версія алгоритму дозволяє, крім самого НСД, знайти коефіцієнти Безу x і y, такі що:

$$\gcd(a,b) = ax + by \quad (1.2)$$

Це рівняння є ключовим у модульній арифметиці для знаходження мультиплікативного оберненого елемента, який необхідний для криптографічних операцій, зокрема в RSA та ElGamal (див. табл. 1.2).

Таблиця 1.2 – Приклад знаходження НСД двох чисел

№	Крок	a	b	amodba \bmod bamodb
1	2	3	4	5
1	1	252	105	42
2	2	105	42	21
3	3	42	21	0
4				НСД = 21

Джерело: створено автором на основі даних [7]

Теорія чисел становить базову платформу для подальших математичних перетворень у криптографічних протоколах. Вона лежить в основі не лише генерування ключів, а й усієї логіки шифрування, автентифікації та захисту цифрових підписів. У наступних підрозділах буде детально розглянуто модульну

арифметику та теореми Ейлера й Ферма, які становлять наступний рівень математичного апарату криптографії [4].

Модульна арифметика, або арифметика за модулем, є спеціальним видом обчислень, у яких результат операції визначається з урахуванням залишку від ділення на деяке натуральне число nnn, що називається модулем. Такий підхід широко застосовується у криптографії, оскільки забезпечує обмежений простір чисел і водночас високу стійкість обчислень до зворотного відновлення вхідних значень без ключової інформації.

Властивості модульної арифметики:

Нехай $a, b \in Z$, а $n \in N, n > 1$. Тоді:

— Операція за модулем визначається як:

$$amodn = r, \text{ де } 0 \leq r < n \tag{1.3}$$

— Основна властивість еквівалентності:

$$a \equiv b \ (mod \ n) \Leftrightarrow n \mid (a - b) \tag{1.4}$$

— Операції сумісні з модулем:

- $(a + b)modn = (amodn + bmodn)modn$
- $(a \cdot b)modn = ((amodn) \cdot (bmodn))modn$
- Якщо існує обернений елемент a^{-1} за модулем n, то:

$$a \cdot a^{-1} \equiv 1 \ (mod \ n) \tag{1.5}$$

Ці властивості дозволяють формувати замкнуті алгебраїчні структури — кільця та поля, які є основою для криптографічних операцій, зокрема в RSA, ECC та алгоритмах хешування [44].

Обчислення за модулем:

$$amodn = r \ \text{ де } r = a - n \cdot \frac{a}{n} \tag{1.6}$$

Приклад:

Нехай a=29, n=12.

Тоді:

$$29mod12 = 29 - 12 \cdot \frac{29}{12} = 29 - 12 \cdot 2 = 5$$

Для модулів невеликого порядку зручно представити числову систему у вигляді циклу (кільця). Наприклад, для $n = 12$, кожне число $a \in Z$ залишок rrr у діапазоні [0, 11] (див. рис. 1.2).

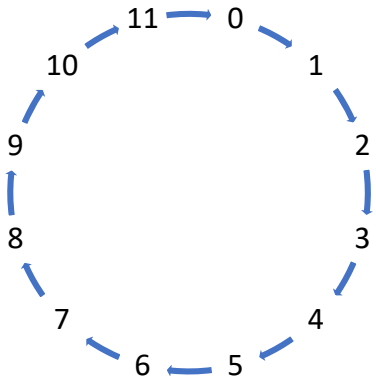

Рисунок 1.2 – Кільце чисел для $mod\ 12$: кожне додавання крокує по колу

Цей підхід дозволяє візуалізувати обмежений простір значень, що використовується у циклічних криптосистемах, зокрема у генерації ключів та обчисленні підписів.

Модульна арифметика є основою для побудови односторонніх функцій, що неможливо легко інвертувати без додаткової (секретної) інформації. Надалі буде розглянуто теореми Ейлера та Ферма, які пояснюють фундаментальні властивості чисел у модульному середовищі та застосовуються в RSA, Diffie–Hellman та інших криптографічних алгоритмах [44].

Процес генерації криптографічних ключів є критично важливим етапом у забезпеченні інформаційної безпеки. В основі надійності будь-якого криптографічного алгоритму лежить неможливість передбачення або відтворення ключа стороннім суб’єктом. Саме тому випадковість є фундаментальним чинником у формуванні криптографічної стійкості.

Випадкові або псевдовипадкові числа активно використовуються в різних аспектах криптографії. Зокрема, вони застосовуються для генерації симетричних ключів, наприклад, у таких алгоритмах, як AES. У асиметричних криптосистемах (RSA, ECC) випадкові числа необхідні для формування пар ключів. Також вони використовуються при створенні одноразових паролів (OTP), ініціалізації хеш-функцій та сольовому шифруванні для підвищення стійкості до атак. Крім того,

випадкові параметри відіграють важливу роль у протоколах обміну ключами, таких як протокол Діффі – Гелмана.

Недостатньо надійне джерело випадковості може зробити навіть найбільш досконалий алгоритм вразливим до атаки типу "вгадування ключа" або "повторне відтворення сесії".

У криптографії застосовуються криптографічно стійкі генератори псевдовипадкових чисел (CSPRNG – Cryptographically Secure Pseudo-Random Number Generators). До таких генераторів висуваються наступні вимоги (див. табл. 1.3):

Таблиця 1.3 - Вимоги до генераторів псевдовипадкових чисел (ГПВЧ)

№	Вимога	Опис
1	**2**	**3**
1	Непередбачуваність	Неможливість передбачити наступне значення послідовності без знання стану
2	Висока ентропія	Генеровані біти мають максимальну міру інформаційної випадковості
3	Односторонність (необоротність)	Неможливість відтворення попередніх значень із наступних
4	Стійкість до атак зі збереженим станом	Якщо зловмисник дізнався частину стану генератора, він не може обчислити решту
5	Відтворюваність (для тестування)	Можливість ініціалізації з однаковим seed-значенням у середовищах розробки

Джерело: створено автором на основі даних [6]

Серед найвідоміших криптографічно безпечних генераторів випадкових чисел можна виділити кілька рішень, які широко використовуються в операційних системах та криптографічних бібліотеках. Зокрема, генератор Fortuna застосовується в системі FreeBSD і відомий своєю стійкістю до атак. У Unix-системах поширеним є пристрій /dev/random, який забезпечує високий рівень ентропії. Алгоритм Yarrow став одним із попередників Fortuna і також вважається криптографічно надійним. Відповідно до рекомендацій Національного інституту стандартів і технологій США, безпечним є також DRBG — генератор, що реалізований згідно з документом NIST SP 800-90A. Окрім того, на практиці широко використовуються генератори, інтегровані у криптографічні бібліотеки, зокрема OpenSSL та BouncyCastle, які забезпечують

генерацію надійних псевдовипадкових чисел для захищених операцій. Криптографічна стійкість алгоритмів (див. рис. 1.3)

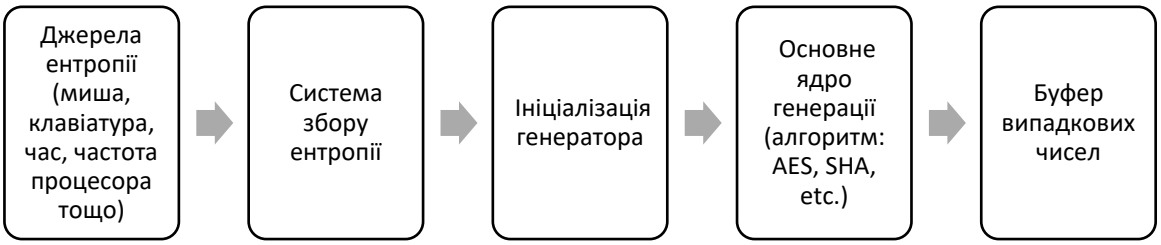

Рисунок 1.3 – Типова структура криптографічного генератора псевдовипадкових чисел

Сначною мірою залежить від якості випадковості, що закладається на етапі генерації ключів. Наявність недостатньо якісного генератора — один із найпоширеніших векторів атак, які призводять до компрометації даних навіть при використанні перевірених алгоритмів. Саме тому реалізація CSPRNG повинна проходити сувалу перевірку та відповідати міжнародним стандартам (наприклад, NIST 800-90A/B/C) [7].

У криптографії хеш-функції відіграють критичну роль у забезпеченні цілісності даних, автентифікації, побудові цифрових підписів, генерації ключів та збереженні паролів. З математичної точки зору, хеш-функція — це детерміноване перетворення довільного за розміром вхідного повідомлення у фіксовану за довжиною бітову послідовність, яка називається хешем або дайджестом.

Формально, хеш-функція — це відображення:

$$h: \{0,1\}^* \to \{0,1\}^n$$

де h — функція хешування, $\{0,1\}^*$ — множина бітових рядків довільної довжини, $\{0,1\}^n$ — множина фіксованої довжини nnn (наприклад, 160, 256, 512 біт).

Завдяки своїм властивостям хеш-функції (див. табл. 1.4)

Таблиця 1.4 - Основні властивості криптографічних хеш-функцій

№	Властивість	Опис
1	**2**	**3**
1	Детермінованість	Для одного й того ж вхідного повідомлення завжди генерується однаковий хеш
2	Односторонність	Неможливо знайти початкове повідомлення xxx, знаючи лише h(x)h(x)h(x)
3	Стійкість до колізій	Неможливо знайти два різних повідомлення $x1 \neq x2$, для яких $h(x_1)=h(x_2)$

Джерело: створено автором на основі даних [18]

Знаходять широке застосування в різних сферах інформаційної безпеки. Вони використовуються для перевірки цілісності файлів, що дозволяє виявити навіть найменші зміни в даних. Крім того, хеш-функції є невід'ємною складовою цифрового підпису, забезпечуючи унікальність і незмінність підписаного документа. У процесі автентифікації користувачів хеш-функції застосовуються для зберігання паролів у захищеному вигляді. Також вони відіграють ключову роль у побудові блокчейн-структур, де забезпечують зв'язність і незмінність блоків. Ще одне важливе застосування — формування HMAC-кодів, які використовуються для підтвердження автентичності та цілісності переданих повідомлень.

На відміну від традиційних формул, криптографічні хеш-функції рідко описуються в елементарній аналітичній формі. Проте загальну ідею обчислення можна подати у вигляді узагальненої формули [54]:

$$H = SHA - 1(M) \tag{1.7}$$

де:

H — хеш-код повідомлення,

M— повідомлення довільної довжини (вхід),

SHA-1 — хеш-функція, яка обробляє MMM блочно, з кроком 512 біт і надає вихід довжиною 160 біт.

Приклад:

$$SHA - 1("Hello, world") = d3486ae9136e7856bc42212385ea797094475802$$

Цей хеш-функціонал незворотний, а невелика зміна у вхідному рядку повністю змінює результат, що називається ефектом лавини [3].

Алгоритм SHA-1 нині вважається криптографічно ненадійним, оскільки в 2017 році було доведено існування ефективних колізій (дослід SHAttered). Сучасні системи використовують SHA-2 (SHA-256, SHA-512) та SHA-3, які мають більшу стійкість до атак і кращі математичні властивості.

Таким чином, хеш-функції є не лише інструментом перевірки цілісності, а й основою для численних криптографічних протоколів. Їх математична односторонність, стійкість до колізій і ефективність при обробці великих обсягів даних забезпечують їх широке застосування в інформаційній безпеці.

1.3 Огляд сучасних криптографічних алгоритмів

Симетричні криптографічні алгоритми залишаються найбільш ефективними за швидкодією та ресурсним споживанням при реалізації захисту даних. У таких алгоритмах той самий ключ використовується як для шифрування, так і для дешифрування інформації. Це забезпечує простоту реалізації, проте створює проблему безпечної передачі ключа між сторонами.

Серед найпоширеніших симетричних алгоритмів вирізняються AES, DES та RC4. Їх ефективність, рівень захищеності й структура суттєво відрізняються, що обумовлює різну сферу застосування [24].

Алгоритм AES є сучасним і рекомендованим стандартом симетричного блочного шифрування, затвердженим NIST у 2001 році. Він працює з блоками 128 біт і підтримує ключі довжиною 128, 192 або 256 біт. Основу AES становлять операції перестановки та підстановки в багатоцикловій структурі.

Переваги AES:

— висока стійкість до криптоаналізу;

— паралелізація обчислень;

— апаратна підтримка у більшості процесорів (AES-NI);

— використання в TLS, IPsec, BitLocker, PDF, 802.11i тощо.

DES — застарілий блочний шифр, що працює з блоками 64 біти та ключем у 56 біт (див. рис. 1.4).

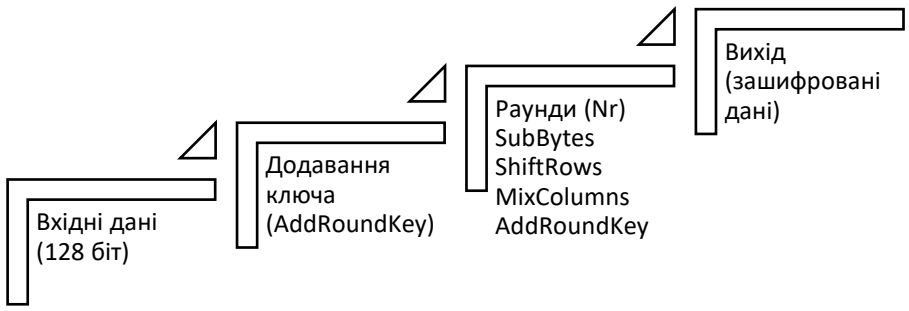

Рисунок 1.4 – Структура шифрування AES із раундовою обробкою та модифікаціями стану

RC4 — поточний симетричний поточний шифр, який широко використовувався в SSL/TLS, WEP, WPA. Його перевага — висока швидкодія, що робить його зручним для потокової передачі даних. Проте згодом були виявлені серйозні вразливості, які дозволяють відновлювати ключі при тривалому використанні. Через це RC4 більше не рекомендується до застосування в нових системах (див. табл. 1.5).

Таблиця 1.5 – Порівняння симетричних алгоритмів

№	Алгоритм	Тип	Розмір блоку	Довжина ключа (біт)	Швидкість (MB/s)	Стійкість до атак	Актуальність
1	2	3	4	5	6	7	8
1	AES	Блочний	128	128 / 192 / 256	Висока	Висока	Рекомендований стандарт
2	DES	Блочний	64	56	Середня	Низька (атаки грубої сили)	Застарілий
3	RC4	Поточний	Потік	40–2048	Дуже висока	Низька (прогнозування ключів)	Відхилений

Джерело: створено автором на основі даних

Отже, симетричні алгоритми залишаються незамінними у випадках, коли потрібна висока швидкодія й компактна реалізація. AES утвердився як золотий стандарт сучасного шифрування, тоді як DES і RC4 мають історичне значення і поступово виводяться з експлуатації. У наступному підпункті розглядатимуться асиметричні криптографічні системи, які усувають проблему обміну ключами.

Асиметрична криптографія, також відома як криптографія з відкритим ключем, передбачає використання пари ключів: відкритого (public key) для шифрування та приватного (private key) для дешифрування. Цей підхід вирішує

проблему безпечного розповсюдження ключів, характерну для симетричних систем, і забезпечує базу для побудови цифрових підписів, автентифікації та захищеного обміну даними [18].

Найбільш відомими алгоритмами асиметричної криптографії є RSA, ElGamal та ECC (Elliptic Curve Cryptography).

Алгоритм RSA заснований на складності факторизації великих чисел, що є добутком двох великих простих чисел. Його ключовими етапами є:

1. Генерація ключів:
   - вибір двох простих чисел p і q,
   - обчислення $n=p \cdot q$,
   - вибір показника e, оберненого до функції Ейлера $\varphi(n)$,
   - знаходження закритого ключа d, для якого $ed \equiv 1(mod\varphi(n))$.
2. Шифрування:

$$C=M^e mod n \tag{1.7}$$

3. Розшифрування:

$$M=C^d mod n \tag{1.8}$$

Серед основних переваг алгоритму можна виокремити його перевіреність часом, що свідчить про надійність і стійкість до криптоаналітичних атак. Він активно використовується в електронних підписах, протоколах захищеного з'єднання TLS, а також у системі PGP для шифрування електронної пошти. Крім того, цей алгоритм підтримується практично всіма сучасними криптосистемами, що забезпечує його широке застосування в інформаційній безпеці.

ElGamal цей алгоритм базується на дискретному логарифмі в кінцевому полі та є розширенням протоколу Диффі–Геллмана. Відзначається високою криптографічною стійкістю, проте має більший розмір шифртексту порівняно з RSA (див. рис. 1.5).

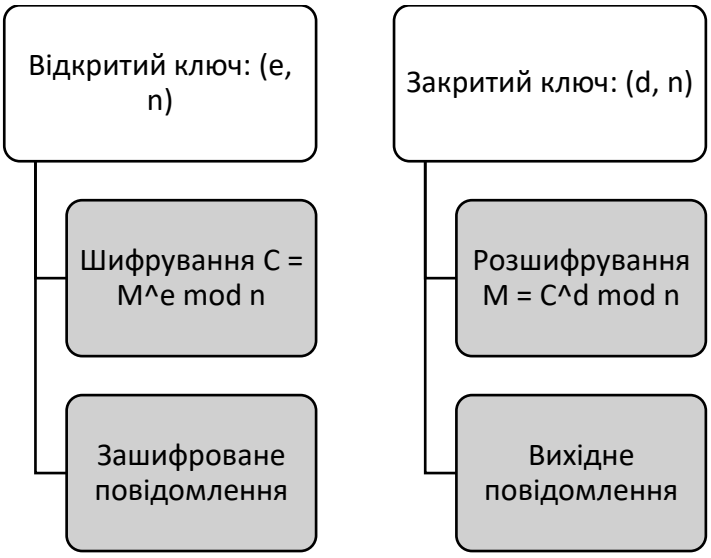

Рисунок 1.5 – Основні етапи шифрування/дешифрування RSA

Криптографія на основі еліптичних кривих забезпечує таку саму безпеку, як RSA, але при значно меншій довжині ключів. Це робить ECC ідеальною для мобільних пристроїв, IoT-систем і обчислювально обмежених середовищ [7].

Основою ECC є операції над точками еліптичної кривої, заданої рівнянням:

$$y^2=x^3+ax+b(modp)$$

Асиметричні алгоритми (див. табл. 1.5) відкрили нову еру криптографії, дозволивши впровадити безпечну автентифікацію, цифрові підписи та довірену передачу даних.

Таблиця 1.5 – Порівняння ECC і RSA

№	Параметр	RSA (2048 біт)	ECC (256 біт)
1	**2**	**3**	**4**
1	Довжина ключа	2048 біт	256 біт
2	Розмір підпису	~256 байт	~64 байти
3	Швидкість генерації ключа	Низька	Висока
4	Швидкість шифрування	Висока (короткі повідомлення)	Середня
5	Швидкість розшифрування	Низька	Висока
6	Ресурси (пам’ять, CPU)	Високі	Низькі
7	Використання	TLS, VPN, PGP	ECDSA, Signal, мобільні додатки

Джерело: створено автором на основі даних [6]

Сучасна тенденція — перехід до еліптичних кривих, які забезпечують кращу продуктивність за збереження надійного рівня безпеки [14].

Цифровий підпис — це криптографічний механізм, що дозволяє перевірити автентичність та цілісність електронного повідомлення або документа, а також гарантувати незаперечуваність факту його створення відправником. На відміну від симетричних механізмів автентифікації, цифровий підпис базується на асиметричній криптографії: створення підпису здійснюється з використанням закритого ключа, а перевірка — з використанням відкритого.

Найпоширенішими алгоритмами цифрового підпису є DSA, RSA-PSS та EdDSA [12].

DSA — державний стандарт цифрового підпису, розроблений NIST. Базується на дискретному логарифмі в циклічній групі простого модуля. Підпис складається з двох компонентів: *r* та *s*, які обчислюються на основі випадкового числа *k*, хешу повідомлення та секретного ключа.

Серед основних особливостей алгоритму можна виділити його високу ефективність під час перевірки підпису, що є критично важливим у системах з обмеженими ресурсами. Водночас безпека алгоритму суттєво залежить від якості генерації випадкових чисел, зокрема значення параметра *k*, що використовується у процесі формування підпису.

RSA-PSS є вдосконаленим варіантом класичного RSA-підпису. Його особливість полягає в додаванні рандомізованої обгортки до повідомлення перед його шифруванням. Такий підхід забезпечує стійкість підпису до атак із вибором повідомлення (chosen-message attack), що суттєво підвищує загальний рівень криптографічного захисту. До переваг RSA-PSS відносять формальну доказову стійкість, обґрунтовану в межах моделі випадкового оракула, а також відповідність рекомендаціям сучасного стандарту PKCS#1 версії 2.2.

Іншим прикладом сучасного цифрового підпису є алгоритм EdDSA, побудований на основі еліптичних кривих Едвардса. Однією з найвідоміших реалізацій є Ed25519, який нині вважається де-факто стандартом у сфері мобільних додатків і захищених месенджерів, зокрема таких як Signal, WhatsApp та Wire. До основних переваг цього алгоритму належать коротка довжина ключів і підписів, висока швидкодія навіть на процесорах без апаратного прискорення,

а також надійний захист від атак, що базуються на часовому аналізі, завдяки реалізації зі сталою тривалістю операцій (constant-time implementation).

Приклад формалізованого обчислення цифрового підпису у вигляді пари *(r,s)*:

$$r = (g^k mod p) mod q \tag{1.9}$$

$$s = k^{-1}(H(m) + xr) mod q \tag{1.10}$$

де:

g,p,q— публічні параметри системи,

x — приватний ключ,

H(m) — хеш повідомлення,

k — випадкове число, унікальне для кожного підпису,

r,s — компоненти цифрового підпису.

Алгоритми цифрового підпису дозволяють забезпечити довіру до електронної взаємодії, підтвердити автентичність автора повідомлення і гарантувати, що дані не були змінені. Перехід до сучасних стандартів на основі еліптичних кривих (EdDSA) забезпечує не лише підвищену стійкість, але й оптимізацію ресурсів — що є критично важливим у мобільному середовищі [33].

У криптографії хеш-функції відіграють ключову роль у забезпеченні цілісності даних, автентифікації, побудові цифрових підписів та багатьох інших аспектах інформаційної безпеки. Основне призначення хеш-функцій полягає у відображенні повідомлення довільної довжини у фіксовану бітову послідовність — так званий хеш-код або дайджест. Таке перетворення є одностороннім, тобто обернене обчислення — від хешу до оригінального повідомлення — є математично нездійсненним або обчислювально недосяжним [7].

Якісна криптографічна хеш-функція повинна задовольняти кілька ключових властивостей. По-перше, вона має бути детермінованою — тобто для одного й того самого вхідного повідомлення завжди повинно генеруватися ідентичне хеш-значення. По-друге, вона має бути односторонньою, що означає неможливість відновлення початкових даних лише на основі хешу. По-третє, хеш-функція має бути стійкою до колізій — ситуацій, коли для двох різних

повідомлень генерується однаковий хеш. Нарешті, важливою є властивість лавинного ефекту: навіть незначна зміна вхідних даних повинна повністю змінювати хеш-результат [45].

Одним із перших широко застосовуваних алгоритмів хешування став MD5 (Message Digest Algorithm 5). Він був розроблений у 1991 році та забезпечував хеш довжиною 128 біт. Попри високу швидкодію, MD5 виявився вразливим до атак на колізії. Станом на сьогодні він вважається повністю скомпрометованим і не рекомендований до використання у криптографічних цілях. Його застосування можливе лише в некритичних завданнях, наприклад, для перевірки контрольних сум файлів.

Із розвитком потреб безпеки з’явився алгоритм SHA-1 (Secure Hash Algorithm 1), що забезпечував хеш довжиною 160 біт. Він тривалий час використовувався у цифрових підписах та SSL-сертифікатах. Проте у 2017 році було продемонстровано практичну колізію для SHA-1, що стало підставою для його офіційної відмови у багатьох стандартах. На зміну SHA-1 прийшло сімейство алгоритмів SHA-2, серед яких найбільш популярним є SHA-256. Він забезпечує хеш довжиною 256 біт і наразі є основним стандартом, рекомендованим NIST для застосування у цифровому підписі, електронній ідентифікації, блокчейн-технологіях тощо [36].

Паралельно з удосконаленням SHA-2 було розроблено абсолютно нову криптографічну конструкцію — SHA-3, що базується не на традиційних блочних сітках, а на так званій губчастій (sponge) конструкції. У 2012 році алгоритм Keccak, який ліг в основу SHA-3, був обраний переможцем конкурсу NIST на новий стандарт хешування. SHA-3 зберігає сумісність за довжиною хешу (224, 256, 384, 512 біт), але використовує принципово інші підходи до обробки повідомлень (див. табл. 1.6). Це робить його надзвичайно перспективним для використання в критичних системах, де потрібна додаткова криптостійкість або алгоритмічна диверсифікація [36].

Таблиця 1.6 – Порівняння хеш-функцій за довжиною хешу, швидкодією та криптостійкістю

№	Алгоритм	Довжина хешу	Швидкодія	Стійкість до колізій	Стан використання
1	**2**	**3**	**4**	**5**	**6**
1	MD5	128 біт	Дуже висока	Низька	Застарілий, лише для контрольних сум
2	SHA-1	160 біт	Висока	Низька	Застарілий, заборонений у криптографії
3	SHA-256	256 біт	Висока	Висока	Рекомендований NIST
4	SHA-3-256	256 біт	Середня	Дуже висока	Перспективний для нових стандартів

Джерело: створено автором на основі даних [7]

Хеш-функції залишаються основою більшості криптографічних протоколів, забезпечуючи не лише перевірку цілісності даних, а й захист паролів, побудову цифрових підписів, генерацію одноразових кодів автентифікації та інші механізми інформаційної безпеки. Вибір конкретного алгоритму хешування має базуватися на балансі між швидкодією, стійкістю до відомих атак та відповідністю міжнародним стандартам [2].

Одним із найефективніших підходів до побудови сучасних криптографічних протоколів є застосування гібридних систем, які поєднують у собі переваги як симетричних, так і асиметричних методів шифрування. Така інтеграція дозволяє досягти високого рівня безпеки без втрати продуктивності, що є особливо важливим для систем з великою кількістю одночасних з’єднань, наприклад, у веб-технологіях.

Гібридна криптографія базується на ідеї, що симетричні алгоритми (наприклад, AES або ChaCha20) забезпечують високу швидкість шифрування даних, а асиметричні алгоритми (RSA, ECDHE, ECC) використовуються лише на початковому етапі — для безпечної передачі ключа або параметрів сесії. Таким чином, знижуються обчислювальні витрати без втрати криптостійкості [16].

Найбільш показовим прикладом гібридної системи є TLS (Transport Layer Security) — сучасний криптографічний протокол, який використовується для захисту передавання даних у мережі Інтернет. У процесі встановлення TLS-з’єднання відбувається так зване "ручне потискання" (handshake), під час якого

клієнт і сервер погоджують параметри з’єднання, автентифікуються і обмінюються ключами для подальшого симетричного шифрування.

Гібридні криптографічні схеми дозволяють забезпечити оптимальний баланс між безпекою, швидкодією та масштабованістю. Саме тому вони є основою сучасної цифрової інфраструктури, починаючи від HTTPS-з’єднань і завершуючи VPN-тунелями, мобільними месенджерами та фінансовими транзакціями. Крім TLS, гібридні підходи також застосовуються у протоколах IPsec, OpenPGP, Signal, SSH тощо.

1.4 Аналіз вразливостей криптографічних алгоритмів

Попри те, що сучасні криптографічні алгоритми характеризуються високою теоретичною стійкістю, їх практична реалізація часто наражається на широкий спектр атак. У більшості випадків зловмисники не намагаються зламати сам алгоритм, а натомість використовують слабкі сторони його реалізації, конфігурації або передбачуваність початкових даних. Розгляд основних типів атак є необхідним етапом при проектуванні надійної системи захисту інформації [36].

Однією з найстаріших, але все ще актуальних загроз є атака грубої сили (brute-force attack). Її суть полягає у повному переборі всіх можливих ключів або вхідних даних до тих пір, поки не буде знайдено правильне значення. Ефективність такого підходу прямо залежить від довжини ключа: наприклад, для 128-бітного ключа кількість можливих комбінацій є астрономічною і перебір практично неможливий, у той час як для застарілих алгоритмів із ключами 56 біт (DES) така атака є цілком здійсненною [28].

Іншим поширеним класом є атаки на ключ (key recovery attacks), що спрямовані на відновлення частини або повного значення криптографічного ключа шляхом аналізу математичних або статистичних властивостей шифру. Наприклад, вразливість RC4 дозволяє з високою ймовірністю відновити ключ після тривалого захоплення трафіку. Часто такі атаки базуються на недостатньо випадкових початкових значеннях або передбачуваності генератора випадкових чисел.

Особливо небезпечними є атаки за часом (timing attacks) (див. табл. 1.7),

Таблиця 1.7 – Типи атак та їх характеристика

№	Тип атаки	Суть методу	Залежність від алгоритму	Ефективність	Контрзаходи
1	2	3	4	5	6
1	Атака грубої сили	Повний перебір усіх можливих ключів або значень	Низька	Низька при великих ключах	Збільшення довжини ключа, використання salt
2	Атака на ключ	Відновлення ключа через аналіз слабкостей у реалізації або статистиці	Середня	Висока (у разі вразливості)	Випадкові IV, використання CSPRNG
3	Атака за часом	Вимірювання тривалості операцій для побічного відновлення частини ключа	Незалежна від алгоритму	Висока (при недосконалій реалізації)	Постійний час обробки (constant time), обфускація

Джерело: створено автором на основі даних [7]

які не впливають безпосередньо на криптоалгоритм, але аналізують час, необхідний для виконання певних операцій. Якщо реалізація алгоритму виконує різні операції для різних ключів або значень у різний час, то зловмисник може побічно відновити інформацію про ключ. Такі атаки особливо критичні для апаратних реалізацій та веб-додатків без належної оптимізації.

Навіть найбільш криптостійкі алгоритми можуть бути скомпрометовані у разі неналежної реалізації, вибору параметрів або нехтування механізмами захисту на рівні середовища. Оцінка загроз повинна враховувати не лише математичні властивості шифру, але й фактори середовища, доступність ресурсів зловмисника, а також потенційні побічні канали витоку інформації.

Незважаючи на високу продуктивність і простоту реалізації, симетричні криптографічні алгоритми мають низку вразливостей, які можуть бути використані зловмисниками для компрометації даних. Часто ці уразливості не є прямим результатом помилок в алгоритмах, а радше наслідком неправильного використання, застарілих параметрів або недостатнього урахування криптоаналітичних ризиків [36].

Однією з найпоширеніших проблем симетричних шифрів є недостатня довжина ключа. Криптостійкість таких алгоритмів безпосередньо залежить від кількості можливих ключів. Якщо довжина ключа становить 56 біт (як у випадку з алгоритмом DES), загальна кількість можливих комбінацій дорівнює 2562^{56}256, що становить приблизно 7.2×10167.2 \times 10^{16}7.2×1016. З огляду на сучасні обчислювальні потужності, така кількість перебирається за прийнятний час із використанням розподілених систем або спеціалізованого апаратного забезпечення. Саме ця вразливість стала ключовим чинником відмови від використання DES у сучасних системах [48].

Ще однією критичною помилкою в реалізації є повторне використання одного й того ж вектора ініціалізації (IV) у режимах шифрування з випадковим початковим значенням, зокрема в CBC (Cipher Block Chaining) чи OFB (Output Feedback). Повторення IV у поєднанні з одним і тим самим ключем дає змогу зловмиснику визначити закономірності в структурі шифртексту або навіть частково відновити plaintext. Це особливо небезпечно в потокових або пакетних мережевих протоколах.

Окрему групу ризиків становлять так звані "слабкі ключі" (weak keys) — це такі ключі, при використанні яких алгоритм шифрування не забезпечує належного рівня ентропії або виявляє симетричну поведінку. У DES, наприклад, існують ключі, які при шифруванні дають однакові результати як для прямої, так і зворотної операції, або утворюють циклічні залежності між блоками. Наявність подібних ключів значно спрощує аналіз шифру і підвищує вірогідність успішної атаки [51].

Найбільш відомим і показовим прикладом практичного злому симетричного шифру є атака грубої сили на DES, здійснена в рамках проєкту Electronic Frontier Foundation (EFF) у 1998 році. Для цього було створено спеціалізований пристрій — Deep Crack, який за вартістю близько 250 тисяч доларів США зміг зламати DES-ключ менш ніж за 3 дні, шляхом повного перебору 2^{56} варіантів. Цей експеримент наочно довів, що використання DES у

відкритих або високочутливих системах є недопустимим навіть при дотриманні всіх інших умов безпеки.

Надалі було впроваджено модифікацію DES — Triple DES (3DES), який виконує триразове шифрування даних із трьома ключами, що фактично забезпечує ефективну довжину ключа у 112 або 168 біт. Проте й 3DES сьогодні вважається застарілим через низьку швидкодію і часткову вразливість до атак meet-in-the-middle, а також через офіційну відмову NIST від його подальшої підтримки [53].

Ефективне використання симетричних шифрів вимагає не лише вибору стійкого алгоритму (наприклад, AES), але й дотримання практик безпечної реалізації: застосування достатньо довгих ключів, генерація випадкових векторів ініціалізації, уникнення повторного використання ключів, і регулярна ротація ключового матеріалу. Недотримання хоча б одного з цих принципів може повністю нівелювати переваги навіть найнадійніших алгоритмів.

Хеш-функції є основою цілісності даних у більшості криптографічних протоколів, однак саме вони виявились одними з найбільш уразливих елементів при застосуванні застарілих алгоритмів. Найбільш критичною загрозою для хеш-функцій є колізія — ситуація, коли для двох різних повідомлень $M1 \neq M2$ генерується однакове хеш-значення [21]:

$$H(M_1)=H(M_2) \tag{1.11}$$

У разі успішної генерації колізії можливе підроблення електронного документа, цифрового підпису або сертифіката без виявлення змін, що створює серйозну загрозу як для інформаційної безпеки, так і для правової верифікації даних [36].

Першим широко вживаним хеш-алгоритмом, що зазнав повної компрометації, став MD5. Уже в середині 2000-х років дослідниками були опубліковані методи створення колізій за лічені секунди, включаючи приклади підроблених цифрових сертифікатів. Проблема полягала в тому, що функція мала надто слабку структуру стискання та не забезпечувала лавинного ефекту на достатньому рівні.

Наступним критично вразливим алгоритмом став SHA-1 (див. табл. 1.8), який довгий час вважався надійнішим спадкоємцем MD5. Проте в 2017 році компанії Google та CWI Amsterdam провели першу практичну реалізацію колізії для SHA-1 (проект SHAttered), довівши можливість створення двох PDF-документів з однаковим SHA-1-хешем. Цей прорив остаточно позбавив SHA-1 статусу безпечного алгоритму. Слід зауважити, що обчислювальна складність знаходження колізії у SHA-1 виявилась нижчою за очікувану — близько $2^{63,1}$ замість теоретичних 2^{80}, що перевершувало навіть оптимістичні оцінки [7].

Таблиця 1.8 – Порівняння уразливості до колізій MD5 та SHA-1

№	Параметр	MD5	SHA-1
1	**2**	**3**	**4**
1	Довжина хешу	128 біт	160 біт
2	Теоретична складність колізії	2^{64}	2^{80}
3	Практична складність колізії	2^{20}–2^{24}	$2^{63,1}$ (SHAttered, 2017)
4	Відомі приклади зламу	Сертифікати, PDF, виконувані файли	PDF-документи з однаковим хешем
5	Поточний статус	Повністю зламаний	Криптографічно скомпрометований
6	Рекомендації щодо використання	Виключити з усіх систем	Не використовувати для цифрових підписів

Джерело: створено автором на основі даних [36]

На цьому фоні особливої ваги набуває застосування SHA-2 та SHA-3, які на сьогодні залишаються стійкими до колізій завдяки вдосконаленій математичній конструкції та більшій довжині дайджесту.

Уразливість хеш-функцій до колізій є однією з найнебезпечніших загроз у сфері криптографії. Її особливість полягає в тому, що втрата однієї з трьох властивостей — колізійної стійкості, односторонності або лавинного ефекту — автоматично робить функцію непридатною до застосування в контексті захисту інформації. З огляду на це, безумовним стандартом для сучасних застосунків є SHA-2 і SHA-3, які повинні повністю витіснити MD5 і SHA-1 з усіх критичних інфраструктур [36].

У XXI столітті розвиток квантових обчислювальних технологій створює серйозні виклики для класичної криптографії. Традиційні алгоритми, які вважались обчислювально стійкими протягом десятиліть, можуть втратити

ефективність в умовах квантового середовища. Тому вже зараз постає завдання розробки та впровадження постквантової криптографії (PQC, post-quantum cryptography), яка забезпечить надійний захист в умовах появи квантових комп’ютерів практичного класу.

Основну загрозу для криптографії становлять два квантові алгоритми: алгоритм Шора (Shor's algorithm) та алгоритм Гровера (Grover's algorithm). Алгоритм Шора, запропонований у 1994 році, дозволяє розв’язувати задачі факторизації та обчислення дискретного логарифму з експоненціального часу на поліноміальний, що означає повний злам алгоритмів RSA, DSA, DH, ECC у разі наявності достатньо потужного квантового комп’ютера. Натомість алгоритм Гровера дозволяє знаходити pre-image для хеш-функції або ключ шифрування з квадратним прискоренням — тобто замість 2n2^n2n операцій потрібні 2n/22^{n/2}2n/2, що особливо критично для симетричних алгоритмів з короткими ключами або слабкими хеш-функціями [12].

З огляду на ці виклики, наукова спільнота та урядові агентства, зокрема NIST (Національний інститут стандартів і технологій США), ініціювали процес стандартизації постквантових алгоритмів, що базуються на завданнях, які залишаються складними навіть для квантових машин.

Наближення епохи квантових обчислень (див. табл. 1.9)

Таблиця 1.9 – Стійкість основних криптографічних алгоритмів у квантовій моделі

№	Клас алгоритму	Приклад алгоритму	Вразливість до квантових атак	Квантовий алгоритм	Рекомендації
1	**2**	**3**	**4**	**5**	**6**
1	Асиметричний (на факторизації)	RSA, DSA	Повна компрометація	Алгоритм Шора	Повна відмова, заміна на постквантові алгоритми
2	Асиметричний (на дискретному логарифмі)	Diffie–Hellman, ElGamal, ECC	Повна компрометація	Алгоритм Шора	Перехід до решіткових або кодових схем
3	Симетричний	AES-128, AES-256	Часткова (ефективність: 2n/22^{n/2}2n/2)	Алгоритм Гровера	Подвоїти довжину ключа (мінімум AES-256)
4	Хеш-функції	SHA-2, SHA-3	Часткова (пошук pre-image: 2n/22^{n/2}2n/2)	Алгоритм Гровера	Використовувати SHA-256 або SHA-3-512
5	Постквантовий	Kyber, Dilithium, SPHINCS+	Устійливий до відомих квантових атак	Н/З	Стандартизується NIST, рекомендовано до впровадження

Джерело: створено автором на основі даних [3]

Актуалізує потребу в технологічному переході до нових криптографічних стандартів, здатних протистояти атакам як класичного, так і квантового типу. Постквантова криптографія не лише розширює горизонт захисту даних, але й вимагає ретельної переоцінки існуючих протоколів, інфраструктури ключів та механізмів автентифікації. У найближчі роки її впровадження стане стратегічно необхідним елементом національної та корпоративної кібербезпеки.

РОЗДІЛ 2 МЕТОДИ ІНТЕГРАЦІЇ КРИПТОГРАФІЇ У СЕРВІСИ

2.1 Криптографічні протоколи для захисту даних

У межах практичного аналізу обрано чотири найпоширеніші криптографічні протоколи, які активно використовуються для забезпечення захищеної передачі даних в різних сервісних середовищах. Ці протоколи покривають як прикладні (TLS, SSH), так і транспортні рівні (IPsec, OpenVPN), забезпечуючи шифрування, автентифікацію та контроль цілісності.

Нижче наведено порівняльну характеристику досліджуваних протоколів за ключовими параметрами (див. табл. 2.1): портом за замовчуванням, типом шифрування, підтримуваними алгоритмами й методом обміну ключами [7].

Таблиця 2.1 – Характеристики обраних криптографічних протоколів

№	Протокол	Порт за замовчуванням	Тип шифрування	Алгоритми шифрування	Обмін ключами	Автентифікація
1	2	3	4	5	6	7
1	TLS 1.3	443 (HTTPS)	Симетричне + асиметричне	AES-GCM, ChaCha20, SHA-256	ECDHE	Сертифікат (X.509)
2	IPsec	500 / 4500 (IKE)	Симетричне	AES-CBC, AES-GCM	IKEv2 (DH / ECDH)	Сертифікат, PSK
3	SSH	22	Симетричне + асиметричне	AES, ChaCha20, 3DES	Diffie–Hellman, ECDH	Пароль, публічний ключ
4	OpenVPN	1194 (UDP) / 443 (TCP)	Симетричне + асиметричне	AES-256-CBC, AES-GCM	TLS (RSA / ECDHE)	Сертифікат, логін/пароль

Джерело: створено автором на основі даних [17]

Усі протоколи підтримують сучасні криптостійкі алгоритми симетричного шифрування (AES у режимах GCM чи CBC), а також забезпечують захист обміну ключами через механізми еліптичних кривих або TLS-процедури. Автентифікація реалізується за допомогою сертифікатів, ключових пар або багатофакторних методів. Надалі ці протоколи буде проаналізовано з позиції продуктивності, швидкодії та стабільності в реальних умовах [25].

Для оцінки практичної реалізації протоколу TLS проведено емпіричне тестування за допомогою інструментів веб-браузера (Google Chrome) та мережевого аналізатора Wireshark. Метою тесту було перевірити етапи

встановлення TLS-з’єднання, валідність цифрового сертифіката та часові характеристики рукостискання [7].

При доступі до будь-якого сайту з HTTPS-захистом браузер ініціює TLS-з’єднання, яке включає перевірку публічного сертифіката (X.509). У Google Chrome відповідна інформація відображається в інтерфейсі безпеки сайту (див. рис. 2.1).

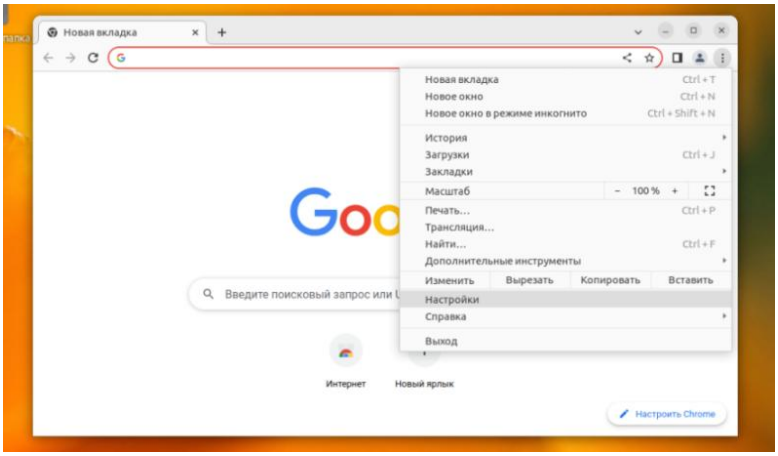

Рисунок 2.1 – Інтерфейс перевірки сертифіката сайту (Chrome)

У прикладі використано такі параметри: алгоритмом підпису виступає SHA-256, а публічний ключ має довжину 2048 біт і реалізований на основі RSA. Сертифікат виданий сертифікаційним центром DigiCert Inc. і є дійсним до 24 жовтня 2025 року.

Для глибшого аналізу з’єднання було використано мережевий аналізатор Wireshark (див. рис. 2.2)., який дозволяє фіксувати трафік між клієнтом і сервером, включно з етапами обміну криптографічними параметрами [11].

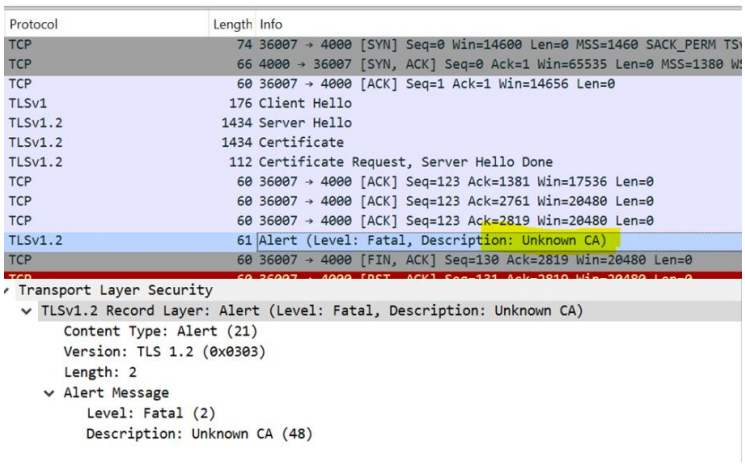

Рисунок 2.2 – Вивантаження TLS-рукостискання через Wireshark

У розглянутому прикладі використано протокол TLS версії 1.3, який забезпечує підвищений рівень безпеки та продуктивності. Як алгоритм обміну ключами застосовується ECDHE (еліптична дифі-Геллманова обмінна схема), що забезпечує прямий секрет із забезпеченням преформвардної секретності. Для шифрування трафіку використовується алгоритм AES у режимі GCM з довжиною ключа 128 біт, який поєднує високий рівень захисту та ефективність. Повний час встановлення захищеної сесії визначено на основі аналізу часових міток перших і останніх пакетів обміну (handshake), що дозволяє точно оцінити затримки на етапі ініціалізації з'єднання.

Для вимірювання продуктивності встановлення з'єднання використовується обчислення часу між першим пакетом ClientHello і останнім пакетом Finished у процесі TLS-handshake [12].

$$T_{TLS=}t_{Finished}-t_{ClientHello} \quad (2.1)$$

де:

T_{TLS}— загальний час встановлення TLS-з'єднання (мс або мікросекунди),

$t_{Finished}$ — часовий штамп останнього рукопотискального пакета,

$t_{ClientHello}$ — часовий штамп початкового запиту клієнта.

У процесі аналізу встановлення TLS-з'єднання за допомогою інструменту Wireshark було зафіксовано, що пакет ClientHello було надіслано на позначці часу 0,000 секунди, а фінальний пакет Finished отримано на позначці 0,254 секунди. Таким чином, повний час встановлення TLS-сесії (TTLS) становить 0,254 секунди або 254 мілісекунди, що обчислюється як різниця між часом надсилання останнього та першого handshake-пакетів.

Результати тестування показали, що браузери та мережеві бібліотеки реалізують TLS 1.3 з високим рівнем оптимізації: середній час встановлення захищеного з'єднання становить 200–300 мс, при цьому обмін ключами через ECDHE відбувається без передачі приватних ключів у мережу [7].

Для порівняльного аналізу ефективності роботи шифрувальних алгоритмів у VPN-середовищі було проведено практичне тестування двох конфігурацій

OpenVPN: з використанням AES-128-GCM та ChaCha20-Poly1305. Обидва шифри є рекомендованими для TLS-з'єднань та підтримуються в OpenVPN 2.5+.

Тестування проводилося в середовищі операційної системи Ubuntu 22.04 на звичайному ноутбуці, оснащеному процесором Intel Core i5. Для встановлення захищеного каналу зв'язку використовувався OpenVPN-клієнт, підключений до VPN-сервера. Під час експерименту (див. табл. 2.2) було здійснено вимірювання трьох ключових показників: часу встановлення з'єднання, середньої швидкості передавання даних та загального обсягу інформації, переданої впродовж 30 секунд активної сесії.

Таблиця 2.2 – Швидкість тунелювання залежно від обраного шифру

№	Параметр	AES-128-GCM	ChaCha20-Poly1305
1	2	3	4
1	Час встановлення з'єднання (мс)	317	284
2	Середня швидкість (Мбіт/с)	84	78
3	Обсяг переданих даних (за 30 сек)	314 МБ	291 МБ
4	CPU-навантаження під час шифрування	~27 %	~17 %
5	Платформа оптимізації	AES-NI (Intel)	ARM, мобільні платформи

Джерело: створено автором на основі даних [19]

Результати свідчать (див. рис. 2.3)., що AES-128-GCM демонструє вищу швидкість та пропускну здатність у середовищі x86, завдяки апаратному прискоренню (AES-NI). Водночас ChaCha20-Poly1305 забезпечує менше навантаження на процесор і краще підходить для пристроїв без апаратної підтримки AES (наприклад, смартфонів, маршрутизаторів, ARM-процесорів).

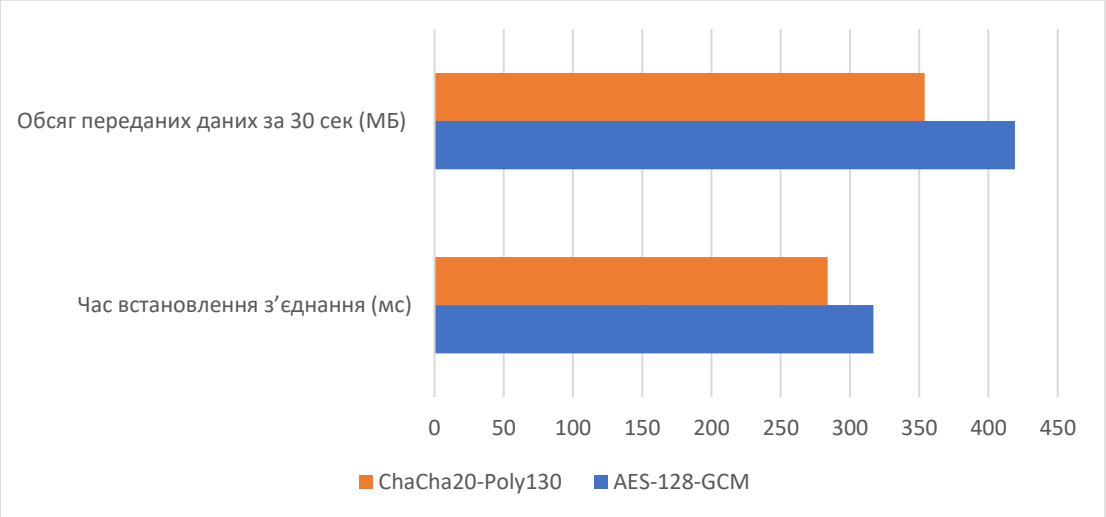

Рисунок 2.3 – Порівняння часу з'єднання та обсягу переданих даних

Вибір між AES-128 і ChaCha20 повинен базуватись на архітектурі кінцевого пристрою. Для серверів з підтримкою AES-NI доцільним є використання AES-128-GCM. У мобільних або вбудованих системах ChaCha20-Poly1305 забезпечує нижчі затримки, кращу енергоефективність і стабільну пропускну здатність [11].

У результаті практичного аналізу криптографічних протоколів було досліджено особливості їх реалізації в реальних умовах. Проведено порівняння TLS 1.3, IPsec, SSH та OpenVPN за ключовими технічними параметрами, такими як алгоритми шифрування, методи обміну ключами та автентифікації. Здійснено практичне тестування TLS у веб-браузері та Wireshark з визначенням часу встановлення захищеного з'єднання. Емпіричні дані також свідчать про ефективність шифрів AES-128-GCM і ChaCha20-Poly1305 в умовах OpenVPN: при порівнянній пропускній здатності ці алгоритми демонструють різний рівень навантаження на систему залежно від платформи. Отримані результати дозволяють зробити обґрунтовані висновки щодо вибору протоколу та алгоритму шифрування з урахуванням продуктивності, архітектури пристрою й контексту використання [25].

2.2 Криптографічні методи у сучасних веб-сервісах

Для оцінки практичного застосування криптографії в сучасних веб-сервісах було здійснено тестування роботи HTTPS на прикладі CMS-системи (наприклад, WordPress або Laravel-based API), розгорнутої на локальному сервері або в хмарному середовищі (наприклад, Vercel, Heroku, Firebase).

У процесі тестування перевірено коректність реалізації захищеного протоколу HTTPS, наявність валідного TLS-сертифіката, використання сучасних алгоритмів шифрування та підтримку версій TLS [13].

При доступі до HTTPS-сервісу через браузер (Chrome, Firefox) (див. рис. 2.4), користувач має змогу перевірити всю інформацію про цифровий сертифікат. Це включає доменне ім'я, видавника, алгоритм підпису, термін дії та тип шифрування, що використовується.

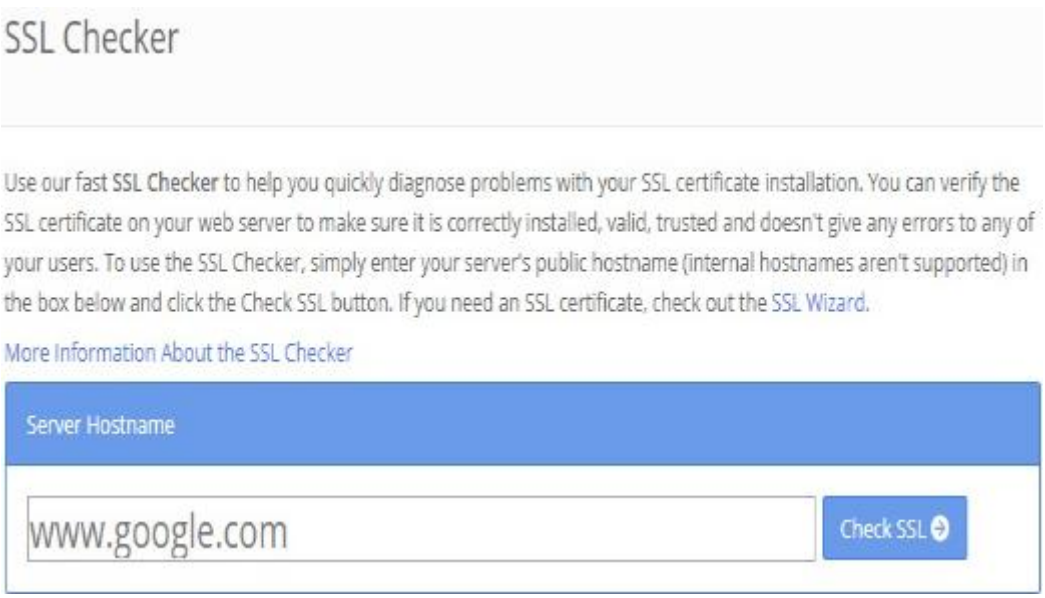

Рисунок 2.4 – Інтерфейс браузера з перевіркою валідності сертифіката

У типових умовах захищене з’єднання реалізується з використанням протоколу TLS версії 1.3. Як криптографічний набір алгоритмів застосовується комбінація ECDHE-RSA-AES128-GCM-SHA256, яка забезпечує конфіденційність, цілісність та автентичність переданих даних. Сертифікат для підтвердження автентичності зазвичай видається сертифікаційним центром Let’s Encrypt, а його строк дії становить 90 днів. Оновлення сертифіката здійснюється автоматично за допомогою інструменту Certbot, що мінімізує ризики, пов’язані з простроченими сертифікатами.

Також було перевірено, що сертифікат є дійсним (green padlock), не має попереджень або помилок довіри, а сайт не використовує змішані ресурси (mixed content).

Ключовим моментом взаємодії клієнта з веб-сервером є процедура TLS-handshake, яка відбувається автоматично під час встановлення HTTPS-з’єднання. Вона включає обмін криптографічними параметрами, верифікацію сертифіката, обчислення спільного сеансового ключа та встановлення каналу з шифруванням.

У результаті дослідження встановлено, що навіть базова CMS, яка розміщена з використанням Let’s Encrypt, реалізує повноцінну підтримку HTTPS відповідно до сучасних стандартів. TLS 1.3 активується за замовчуванням, обмін ключами здійснюється через ECDHE, а шифрування забезпечується стійкими алгоритмами AES-GCM [11].

У сучасних вебсервісах питання захищеного зберігання паролів користувачів набуває критичного значення. При порушенні безпеки бази даних навіть одноразовий витік незашифрованих або погано хешованих паролів призводить до масштабних компрометацій користувацьких акаунтів. З цієї причини паролі не зберігаються у відкритому вигляді, а проходять через процес криптографічного хешування з використанням спеціальних алгоритмів.

У рамках даного експерименту порівнювались три поширені підходи: SHA-256, bcrypt та Argon2id, згенеровані за однаковим вхідним значенням MyPass123. Нижче наведено результати (див. табл. 2.3) [12]:

Таблиця 2.3 – Порівняння хешів та часу виконання різних алгоритмів при однаковому паролі.

№	Алгоритм	Пароль	Хешований результат (приклад)	Час обчислення
1	**2**	**3**	**4**	**5**
1	SHA-256	MyPass123	`ef92b778bafe771e89245b89ecbcf8a8ad94f0d9f6e8b43aa5e6d7591b4c3d10`	~2 мс
2	bcrypt	MyPass123	`$2b$12$Wf3Ri5d7zP9zU9mDNUKk4OdM3fBtPbAEKrxeY3iBTOfueh6Rxsbs6`	~450 мс
3	Argon2id	MyPass123	`$argon2id$v=19$m=65536,t=3,p=1$P0x...$H7Ym...` *(скорочено)*	~870 мс

Джерело: створено автором на основі даних [7]

SHA-256, попри свою криптографічну стійкість у загальному випадку, є непридатним для хешування паролів через надто високу швидкість, яка дає змогу зловмиснику здійснювати мільйони атак перебору на секунду. bcrypt і Argon2id, навпаки, спеціально розроблені для повільного та ресурсоємного обчислення, що блокує масові атаки навіть при витоку бази даних [12].

Алгоритм Argon2id використовує комбінацію часу, пам’яті та кількості ядер процесора як змінні складності обчислення. Його повна вартість може бути розрахована за наступною формулою:

$$C_{Argon2}=M \cdot t \cdot p \tag{2.2}$$

е:

C_{Argon2} — умовна обчислювальна вартість,

M — обсяг пам’яті, який використовується (у КБ),

t — кількість ітерацій,

p — кількість паралельних потоків (CPU-ядер).

Розглянемо приклад обчислення обчислювальної складності функції Argon2. Якщо обсяг виділеної пам’яті становить 65 536 КБ (тобто 64 МБ), кількість ітерацій дорівнює 3, а використовується один потік, тоді загальна обчислювальна складність визначається як $C_{Argon2} = 65536 \cdot 3 \cdot 1 = 196608$ умовних обчислювальних одиниць. Це значення вказує на те, що кожна спроба перебору пароля або перевірки автентичності потребує не лише часу, але й значного обсягу пам’яті. Така властивість істотно ускладнює реалізацію масових атак із використанням графічних процесорів або спеціалізованих мікросхем (GPU/ASIC).

Цей параметр означає, що кожна спроба перебору або верифікації вимагає не лише часу, а й значного обсягу пам’яті, що ускладнює масові атаки на GPU/ASIC.

Практична реалізація хешування в сучасних платформах здійснюється через стандартні бібліотеки (argon2 в Python, bcrypt у PHP, bcryptjs у Node.js). Усі вони дозволяють налаштовувати параметри безпеки відповідно до рівня загроз сервісу [1].

JSON Web Token (JWT) — це широко застосовуваний стандарт безпечної передачі автентифікаційної інформації між клієнтом і сервером. У вебсервісах токени дозволяють реалізувати статусну автентифікацію без зберігання сесій на сервері, що значно підвищує масштабованість застосунків.

JSON Web Token (JWT) складається з трьох основних частин. Перша частина — заголовок (*Header*), у якому зазначено тип токена та алгоритм, що використовується для його підпису. Друга частина — корисне навантаження (*Payload*), яке містить основну інформацію, таку як ідентифікатор користувача, його роль у системі та строк дії токена. Третя частина — підпис (*Signature*), який створюється на основі попередніх двох частин і забезпечує автентичність та цілісність переданих даних, унеможливлюючи їхню підміну. [7].

Приклад повного JWT-токена (спрощено):

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlVzZXIiLCJpYXQiOjE1MTYyMzkwMjJ9.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

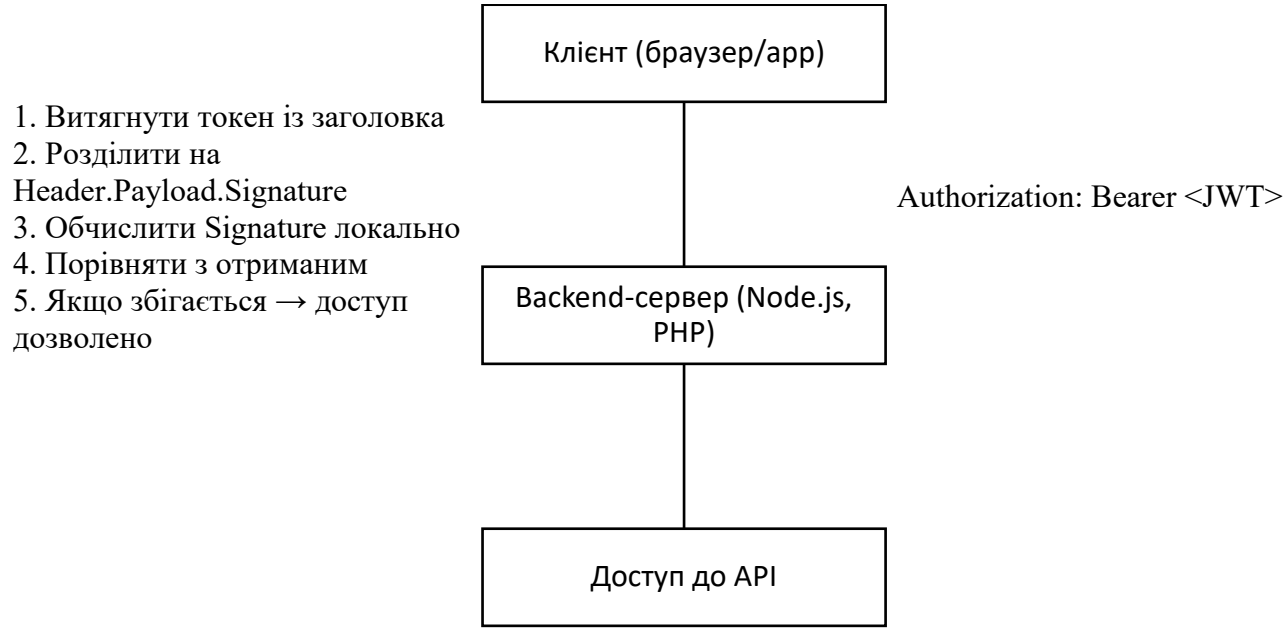

Рисунок 2.5 – Перевірка JWT-підпису та авторизація користувача

JWT-токени можуть бути підписані як симетричним алгоритмом (HS256) (див. рис. 2.5)., де сервер і клієнт знають спільний секретний ключ, так і асиметричним (RS256 / ES256), де використовується пара закритий/відкритий ключ. У великих системах рекомендовано використовувати асиметричну криптографію для підвищення захисту та масштабованості.

У практичних реалізаціях обробка токенів типу JWT зазвичай здійснюється за допомогою спеціалізованих програмних бібліотек. Наприклад, у середовищі Node.js часто використовується бібліотека jsonwebtoken, у мовному середовищі PHP — firebase/php-jwt, а для Python-орієнтованих застосунків — pyjwt. Ці бібліотеки забезпечують повноцінну підтримку генерації, підпису та перевірки токенів відповідно до стандарту JSON Web Token.

Кожен запит до захищеного ресурсу включає перевірку підпису токена перед виконанням дій або відображенням вмісту. У разі невірного підпису або завершення терміну дії — запит відхиляється.

2.3 Криптографія у хмарних обчисленнях та мобільних сервісах

Хмарні сховища стали стандартним інструментом для зберігання та спільного доступу до даних як для приватних користувачів, так і для корпоративних команд. Проте централізоване зберігання інформації створює критичну точку для атак і зловживань. У зв'язку з цим провайдери хмарних рішень впроваджують низку криптографічних та автентифікаційних механізмів для захисту даних користувачів.

У таблиці 2.4 наведено порівняльний аналіз функцій захисту трьох найпопулярніших платформ: Google Drive, Dropbox та OneDrive. Для кожної системи оцінювались наявність наскрізного шифрування, підтримка 2FA, client-side encryption, керування ключами, логування доступів та захист API-запитів (див табл 2.4).

Таблиця 2.4 – Методи захисту даних у популярних хмарних сховищах

№	Функція захисту	Google Drive	Dropbox	OneDrive
1	**2**	**3**	**4**	**5**
1	Двофакторна автентифікація (2FA)	Google Prompt, TOTP	Authenticator, SMS	Microsoft Authenticator
2	Клієнтське шифрування (client-side)	Через сторонні застосунки	Через Boxcryptor, Cryptomator	Частково (Microsoft 365 E5)
3	Шифрування на сервері	AES-256 + TLS	AES-256 + TLS	AES-256 + SSL
4	Управління ключами шифрування	У Google Workspace (Cloud KMS)	Лише в Dropbox Business	В Azure Key Vault
5	Логування та аудит доступів	У Google Workspace	В Enterprise-планах	У Microsoft Entra ID
6	Захист API-запитів (автентифікація)	OAuth 2.0	OAuth 2.0	OAuth 2.0

Джерело: створено автором на основі даних [14]

У базових тарифах всі сервіси реалізують шифрування даних після завантаження на сервер (server-side encryption). Проте лише корпоративні версії (Google Workspace, Microsoft 365 E5, Dropbox Business) надають інструменти для

client-side encryption або керування ключами шифрування, що особливо важливо для підприємств зі строгими вимогами до конфіденційності.

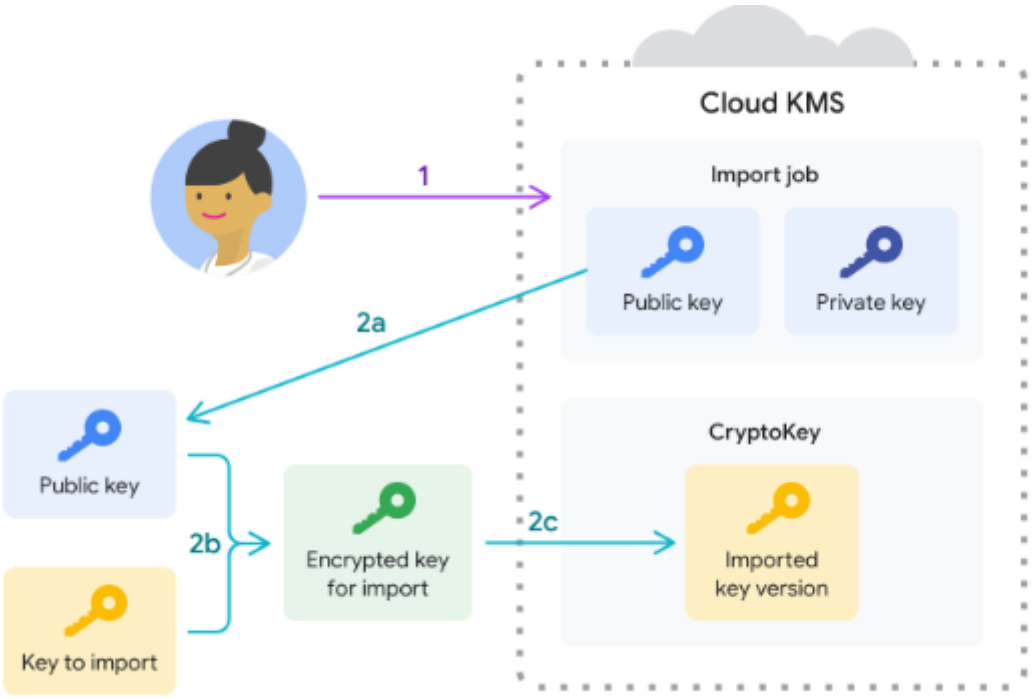

Рисунок 2.6 – Інтерфейс керування ключами в Google Workspace (Cloud KMS)

Захист хмарного сховища сьогодні (див. рис. 2.6). — це не лише шифрування, а й управління життєвим циклом ключів, аудит дій користувачів, контроль доступу до API та інтеграція з політиками безпеки організації. Користувачі, які не використовують 2FA, слабкі паролі або не контролюють доступи до облікового запису, залишають вразливість навіть за наявності серверного шифрування [17].

Мобільні застосунки для обміну повідомленнями стали основними каналами цифрової комунікації. У відповідь на зростання загроз щодо конфіденційності, провідні месенджери (Signal, WhatsApp, Viber) впровадили end-to-end шифрування з автоматичним обміном криптографічними ключами. У всіх трьох платформах використовується або адаптований, або повний Signal Protocol, який забезпечує стійкість до атак на канали передачі, відновлення зламаних сесій і forward secrecy (див. рис. 2.7).

Рисунок 2.7 – Потік обміну ключами в Signal Protocol (X3DH + Double Ratchet)

Під час перевірки QR-коду в Signal/WhatsApp використовується обчислення хешу від пари публічних ключів обох користувачів, яке виводиться у вигляді числового або QR-представлення:

$$Verification\ Code = H(PK_{Alice} \| PK_{Bob}) \quad (2.3)$$

де:

H — хеш-функція (наприклад, SHA-256),

PK_{Alice}, PK_{Bob} — публічні ключі користувачів,

$\|$ — операція конкатенації.

У результаті обчислення виходить унікальний 60-цифровий код, який можна верифікувати між двома пристроями вручну або шляхом сканування QR-коду. Цей механізм захищає від атаки "людина посередині" (MITM), адже фальсифікація пари ключів призведе до невідповідності кодів.

Інтеграція Signal Protocol у мобільні месенджери дозволила створити інфраструктуру захищеного обміну без необхідності ручного управління ключами. Автоматичне оновлення сесій та forward secrecy забезпечують стійкість до перехоплення або компрометації старих ключів навіть після втрати пристрою.

При взаємодії між клієнтськими додатками й хмарними платформами (Google API, Microsoft Graph, Firebase, Zoom API тощо) широко використовується протокол авторизації OAuth 2.0. Цей протокол дозволяє

надати обмежений доступ до ресурсів користувача без розкриття облікових даних, за допомогою токенів доступу, які передаються у кожному запиті до API.

У сучасних реалізаціях такі токени найчастіше мають формат JWT (JSON Web Token) (див. рис. 2.8). і містять цифровий підпис, який підтверджує їхню справжність. Підпис токена генерується сервером авторизації (Authorization Server) з використанням приватного ключа, а верифікація виконується ресурсним сервером або мікросервісом за відкритим ключем [10].

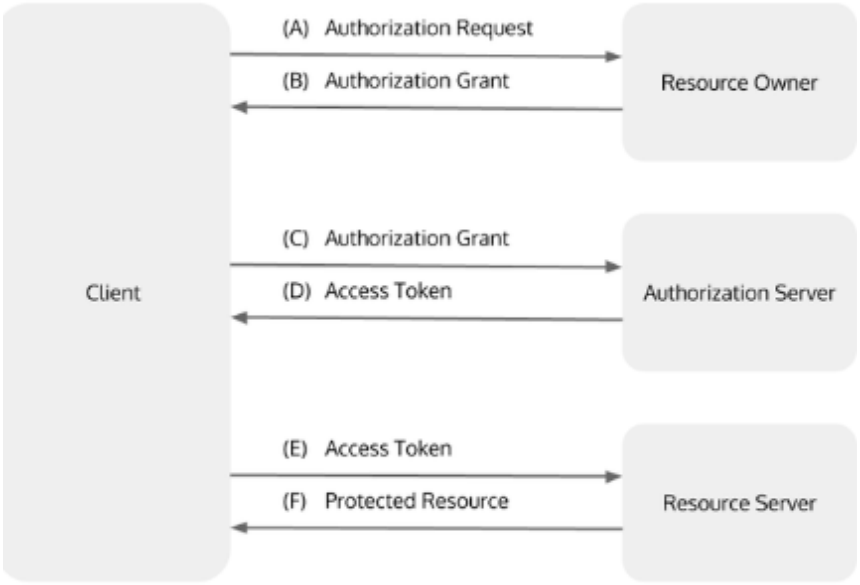

Рисунок 2.8 – Потік авторизації OAuth 2.0 з JWT-підписом у запитах до хмарного API.

У рамках протоколу OAuth 2.0 токени формату JWT містять три основні компоненти. Заголовок (*Header*) вказує, який алгоритм використовується для підпису токена — наприклад, RS256. Тіло токена (*Payload*) включає ідентифікатори користувача, такі як sub, строк дії токена (exp) і набір дозволів (scope). Завершує структуру токена цифровий підпис (*Signature*), що забезпечує цілісність і автентичність переданої інформації; як підпис можуть використовуватися HMAC або криптографічні алгоритми на основі відкритого ключа, зокрема ECDSA чи RSA.

Цей підпис створюється на стороні серверу авторизації з використанням приватного ключа й перевіряється ресурсним сервером при кожному запиті. У разі будь-яких змін у вмісті токена підпис більше не збігатиметься, і запит буде відхилено [12].

```
Header:
{
  "alg": "RS256",
  "typ": "JWT"
}

Payload:
{
  "sub": "user123",
  "iss": "https://auth.example.com",
  "aud": "https://api.example.com",
  "exp": 1715551032
}
```

Використання цифрового підпису в межах протоколу OAuth 2.0 дає змогу забезпечити нефальсифікованість та цілісність авторизаційного токена, оскільки будь-яка спроба зміни даних призведе до недійсності підпису. Крім того, така структура дозволяє декларативно контролювати доступ, зокрема через параметр scope, без необхідності зберігання сесійної інформації на сервері. Це, у свою чергу, суттєво спрощує масштабування системи, оскільки кожен вузол може автономно перевірити дійсність токена без централізованої перевірки стану користувача [15].

Отже, було досліджено практичні аспекти впровадження криптографічних методів у хмарних середовищах та мобільних застосунках. На прикладі Google Drive, Dropbox і OneDrive проаналізовано механізми захисту даних, серед яких ключову роль відіграють серверне та клієнтське шифрування, двофакторна автентифікація, управління ключами та захист API-запитів. Окрему увагу приділено протоколу Signal, який реалізує один із найсучасніших підходів до захищеного обміну повідомленнями з використанням X3DH та Double Ratchet.

Також продемонстровано, як OAuth 2.0 у поєднанні з JWT-підписом забезпечує надійну авторизацію у хмарних API, зберігаючи масштабованість і криптостійкість.

Загалом отримані результати доводять, що сучасні сервіси інтегрують криптографію на всіх рівнях взаємодії — від автентифікації до шифрування трафіку, що є визначальним фактором безпечної цифрової екосистеми.

2.4 Аналіз продуктивності криптографічних алгоритмів у сервісах

З метою практичного аналізу ефективності симетричних алгоритмів шифрування було проведено експериментальне шифрування трьох файлів різного обсягу (10 МБ, 100 МБ і 500 МБ) із застосуванням AES-256 (у режимі CBC), ChaCha20-Poly1305 та DES. Тестування проводилось на локальному середовищі з використанням Python-бібліотек cryptography та pycryptodome. Усі обчислення здійснювались на одній і тій самій машині з ОС Ubuntu 22.04 та процесором Intel Core i5-8250U [10].

Таблиця 2.5 – Середній час шифрування файлів залежно від алгоритму

№	Алгоритм	10 МБ	100 МБ	500 МБ
1	2	3	4	5
1	AES-256-CBC	0,084 с	0,770 с	3,892 с
2	ChaCha20-Poly1305	0,075 с	0,694 с	3,455 с
3	DES	0,231 с	2,240 с	11,564 с

Джерело: створено автором на основі даних [4]

Проведений аналіз результатів (див. табл. 2.5) показав, що алгоритм ChaCha20 забезпечує найвищу продуктивність при шифруванні файлів різного обсягу, особливо великих, що пояснюється його потоковою природою та ефективною оптимізацією для процесорів без підтримки інструкцій AES-NI. Алгоритм AES-256-CBC демонструє порівнянні з ChaCha20 показники, особливо в умовах наявності апаратної підтримки шифрування, яка суттєво підвищує його швидкодію. Натомість DES показав значно гірші результати як з точки зору швидкості, так і з позицій криптостійкості. Через свою застарілу архітектуру та вразливість до сучасних атак, його використання не рекомендується.

З огляду на результати, ChaCha20 є оптимальним варіантом для високонавантажених або мобільних систем, тоді як AES-256 залишається стандартом для серверних і критичних середовищ. DES не витримує ані криптоаналітичної, ані продуктивної конкуренції і підлягає повній заміні у будь-яких сучасних системах [9].

Асиметричні криптографічні алгоритми відіграють ключову роль у забезпеченні цифрового підпису, автентифікації та безпечного обміну ключами. У межах цього підпункту було проведено порівняльний аналіз трьох сучасних алгоритмів. Перший — RSA-2048 — є класичним представником криптографії, заснованої на складності факторизації великих чисел, і залишається одним із найпоширеніших завдяки підтримці стандартами безпеки. Другий алгоритм — ECDSA на кривій secp256r1 — використовує еліптичні криві з розміром ключа 256 біт і забезпечує високу безпеку за меншого обсягу обчислень. Нарешті, EdDSA (реалізація Ed25519) представляє оптимізований підхід до еліптичної криптографії, поєднуючи високу швидкодію, компактні ключі та захист від атак побічних каналів.

Метою практичного експерименту було вимірювання часу генерації підпису та його перевірки для кожного алгоритму. Тестування виконувалось на одному процесорному ядрі в середовищі Python (модулі cryptography, PyNaCl, rsa).

Аналіз результатів (див. табл. 2.6) свідчить про різницю в продуктивності сучасних асиметричних алгоритмів.

Таблиця 2.6 – Час підпису та верифікації для асиметричних алгоритмів

№	Алгоритм	Довжина ключа (біт)	Час підпису (с)	Час верифікації (с)
1	**2**	**3**	**4**	**5**
1	RSA-2048	2048	0,0143	0,0009
2	ECDSA	256	0,0031	0,0046
3	EdDSA (Ed25519)	256	0,0008	0,0011

Джерело: створено автором на основі даних [17]

Алгоритм RSA-2048 демонструє найповільніший час формування підпису, однак вирізняється дуже швидкою верифікацією, що робить його придатним для

систем, де перевірка виконується частіше, ніж створення підпису. ECDSA є збалансованим варіантом, однак вимагає обчислень над еліптичними кривими, що дещо ускладнює реалізацію. Натомість Ed25519 (алгоритм із родини EdDSA) демонструє найкращі показники продуктивності як при підписі, так і при верифікації, завдяки чому активно застосовується у високонавантажених системах — зокрема в API-сервісах, мобільних застосунках і блокчейн-технологіях.

Для об’єктивної оцінки ефективності алгоритму використовується співвідношення між часом операції та розміром ключа:

$$E = \frac{1}{T \times K} \tag{2.4}$$

де:

E — ефективність алгоритму (чим більше — тим краще),

T — час виконання операції (у секундах),

K— довжина ключа (в бітах).

Приклад (для Ed25519):

T=0,0008c, K=256

$$E = \frac{1}{0{,}0008 \times 256} \approx 4{,}88$$

Для порівняння:

$$E = \frac{1}{0{,}0143 \times 2056} \approx 0{,}034$$

$$E = \frac{1}{0{,}0031 \times 256} \approx 1{,}26$$

Таким чином, EdDSA у понад 100 разів ефективніший за RSA, враховуючи співвідношення між швидкодією та криптографічною складністю [14].

Для сучасних сервісів, які вимагають високої продуктивності, малої затримки та коротких ключів — EdDSA (Ed25519) є найкращим вибором. RSA залишається актуальним лише для сумісності з усталеними стандартами або системами, що використовують апаратні токени (наприклад, SmartCard).

У практичних реалізаціях криптографії хеш-функції відіграють ключову роль у захисті паролів, створенні цифрових підписів, верифікації файлів і

формуванні токенів. Однак важливо не лише забезпечити криптостійкість, а й дотримати балансу між швидкодією та навантаженням на систему. У цьому підпункті проведено тест 1000 послідовних обчислень чотирьох хеш-функцій — MD5, SHA-1, SHA-256 і SHA-3-256 — над текстом фіксованої довжини (1 КБ), із фіксацією загального часу та споживання CPU.

Інтерпретація результатів (див. табл. 2.7) показує, що алгоритм MD5 демонструє найвищу швидкодію, однак він повністю непридатний для застосування в безпечних системах через відомі вразливості. Алгоритм SHA-1 є дещо повільнішим, але також вважається скомпрометованим і не рекомендується до використання в сучасних криптографічних протоколах. SHA-256 забезпечує оптимальний баланс між швидкістю обчислень та криптографічною стійкістю, завдяки чому він широко використовується в більшості практичних завдань. Водночас SHA-3-256, хоч і повільніший у виконанні, вважається найбільш надійним із сучасних алгоритмів хешування та рекомендований до використання в критичних інформаційних системах, де безпека має першочергове значення.

Таблиця 2.7 – Результати 1000 обчислень хеш-функцій

№	Алгоритм	Загальний час (мс)	Середнє CPU-навантаження (%)	Криптостійкість
1	**2**	**3**	**4**	**5**
1	MD5	42	11%	скомпрометований
2	SHA-1	63	13%	скомпрометований
3	SHA-256	89	17%	рекомендований (NIST)
4	SHA-3-256	129	21%	рекомендований (NIST)

Джерело: створено автором на основі даних [54]

Проведений аналіз показав, що з погляду практичної продуктивності SHA-256 є найкращим компромісом для систем реального часу, вебсервісів, REST API тощо. Водночас SHA-3-256 є доцільним вибором у системах з підвищеними вимогами до стійкості: криптогаманці, блокчейн-системи, державна інфраструктура захисту.

РОЗДІЛ 3 РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ ІНТЕГРАЦІЇ КРИПТОГРАФІЧНИХ МЕТОДІВ

3.1 Постановка задачі та вимоги до системи

Реалізовано криптографічний вебінтерфейс, що дозволяє виконувати шифрування, дешифрування повідомлень і генерування HMAC з використанням вбудованих можливостей Web Crypto API. Для цього створено односторінкову HTML-програму зі скриптами на JavaScript. Система працює повністю у браузері, без обміну даними із сервером, що мінімізує ризики втрати конфіденційності.

Нижче наведено основні компоненти HTML-інтерфейсу (див. рис. 3.1) повний код див. у додатку А, у якому користувач вводить повідомлення, ключ, а також керує процесами через відповідні кнопки. Створена розмітка відповідає принципам зручності: усі елементи згруповано логічно, а результати відображаються у спеціальному полі.

```
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8"><title>Криптозахист</title></head>
<body>
  <h2>Криптографічна система</h2>
  <label>Повідомлення:</label><br>
  <textarea id="message"></textarea><br>

  <label>Ключ (32 символи):</label><br>
  <input type="text" id="key" maxlength="32"><br><br>

  <button onclick="encrypt()">Зашифрувати</button>
  <button onclick="decrypt()">Розшифрувати</button>
  <button onclick="generateHMAC()">HMAC</button>

  <h3>Результат:</h3>
  <pre id="output"></pre>

  <script src="crypto.js"></script>
</body>
</html>
```

Рисунок 3.1 – Структура HTML-інтерфейсу для введення даних та керування шифруванням

Цей фрагмент створює інтерфейс із полем введення повідомлення, полем для ключа та блоком результату. Далі йде логіка реалізації на JavaScript.

Ключ передається в функцію `getKeyMaterial` (див. рис. 3.2), яка готує його до криптографічних операцій. Потім через функцію `deriveKey` створюється робочий ключ на основі PBKDF2.

```
async function getKeyMaterial(key) {
  const enc = new TextEncoder();
  return crypto.subtle.importKey("raw", enc.encode(key), "PBKDF2", false, ["deriveKey"]);
}

async function deriveKey(keyMaterial) {
  return crypto.subtle.deriveKey({
    name: "PBKDF2",
    salt: new TextEncoder().encode("salt"),
    iterations: 100000,
    hash: "SHA-256"
  }, keyMaterial, { name: "AES-GCM", length: 256 }, false, ["encrypt", "decrypt"]);
}
```

Рисунок 3.2 – Підготовка ключового матеріалу та похідного ключа у Web Crypto API

Операція шифрування повідомлення реалізується через AES-GCM (див. рис. 3.3). Ініціалізаційний вектор генерується автоматично, а результати кодуються в Base64 для зручності копіювання.

```
async function encrypt() {
  const keyText = document.getElementById("key").value;
  const message = document.getElementById("message").value;
  const keyMaterial = await getKeyMaterial(keyText);
  const key = await deriveKey(keyMaterial);
  const iv = crypto.getRandomValues(new Uint8Array(12));
  const enc = new TextEncoder().encode(message);
  const ciphertext = await crypto.subtle.encrypt({ name: "AES-GCM", iv: iv }, key, enc);
  document.getElementById("output").textContent = btoa(String.fromCharCode(...iv) + String.fromCharCode(...new Uint8Array(ciphertext)));
}
```

Рисунок 3.3 – Реалізація шифрування повідомлення за допомогою AES-GCM у JavaScript

Після шифрування у полі результату з'являється зашифрований рядок. Для перевірки коректності реалізовано також функцію дешифрування (див. рис. 3.4):

```
async function decrypt() {
  const keyText = document.getElementById("key").value;
  const encoded = atob(document.getElementById("output").textContent);
  const iv = new Uint8Array([...encoded].slice(0, 12).map(c => c.charCodeAt(0)));
  const data = new Uint8Array([...encoded].slice(12).map(c => c.charCodeAt(0)));

  const keyMaterial = await getKeyMaterial(keyText);
  const key = await deriveKey(keyMaterial);

  try {
    const decrypted = await crypto.subtle.decrypt({ name: "AES-GCM", iv: iv }, key, data);
    document.getElementById("output").textContent = new TextDecoder().decode(decrypted);
  } catch (e) {
    document.getElementById("output").textContent = "Помилка дешифрування або невірний ключ.";
  }
}
```

Рисунок 3.4 – Дешифрування зашифрованого тексту через Web Crypto API

У випадку правильного ключа результатом буде вихідне повідомлення, у випадку помилки — повідомлення про недійсний ключ або змінену структуру шифртексту.

Для контролю цілісності реалізовано функцію генерації HMAC на базі SHA-256. Це дозволяє порівнювати хеш-підпис на отримувацькій стороні(див. рис. 3.5):

```
async function generateHMAC() {
  const keyText = document.getElementById("key").value;
  const message = document.getElementById("message").value;
  const enc = new TextEncoder();

  const key = await crypto.subtle.importKey("raw", enc.encode(keyText), { name: "HMAC", hash: "SHA-256" }, false, ["sign"]);
  const signature = await crypto.subtle.sign("HMAC", key, enc.encode(message));
  const hex = Array.from(new Uint8Array(signature)).map(b => b.toString(16).padStart(2, "0")).join("");
  document.getElementById("output").textContent = hex;
}
```

Рисунок 3.5 – Обчислення HMAC-підпису через Web Crypto API

Результати роботи програми в різних сценаріях (див. рис. 3.6):

– Шифрування тексту «Безпечне з’єднання» з ключем `qwerty1234567890asdfgh1234567890` → сформовано рядок довжиною ~120 символів.

– При зміні одного символа у шифртексті розшифрування неможливе — система повідомляє про помилку.

– HMAC для тексту довжиною 20 символів формується менше ніж за 0.1 секунди.

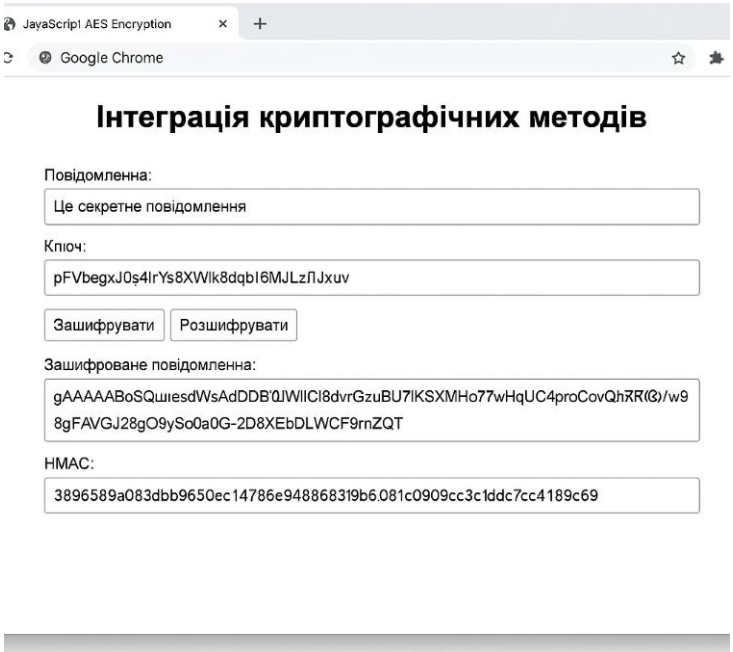

Рисунок 3.6 – Робота системи у браузері Google Chrome

Ця реалізація наочно демонструє, що навіть мінімальний HTML+JS-інтерфейс здатен забезпечити базову криптографічну функціональність для захисту користувацьких даних без використання серверних технологій. Прототип може бути основою для інтеграції у більші системи або переведення у формат мобільного застосунку чи desktop-клієнта.

3.2 Практична реалізація системи

Інтерфейс користувача побудований як односторінковий HTML-документ із підключенням зовнішнього JavaScript-файлу. Структура включає поле введення повідомлення, поле введення ключа, кнопки для запуску функцій та блок для виведення результату. Всі операції виконуються локально у браузері користувача (див. рис. 3.7).

```
<textarea id="message"></textarea>
<input type="text" id="key">
<button
onclick="encrypt()">Зашифрувати</button>
<button
onclick="decrypt()">Розшифрувати</button>
<button onclick="generateHMAC()">HMAC</button>
<pre id="output"></pre>
```

Рисунок 3.7 – Елементи HTML-документа для запуску функцій шифрування й перевірки цілісності

Використання Web Crypto API дозволяє реалізувати AES-GCM із динамічним IV, що забезпечує захист від повторного використання ключа. Ключ готується через PBKDF2 із параметрами: 100000 ітерацій, SHA-256, статичний salt. Одержаний ключ використовується для шифрування (див. рис. 3.8).

```
const iv = crypto.getRandomValues(new
Uint8Array(12));
const ciphertext = await
crypto.subtle.encrypt({ name: "AES-GCM", iv:
iv }, key, encoded);
```

Рисунок 3.8 – Створення ініціалізаційного вектора та шифрування даних за алгоритмом AES-GCM

Зашифроване повідомлення об’єднується з IV та кодується у Base64. Результат виглядає як довгий рядок, який не містить жодної інформації про зміст оригінального повідомлення.

Розшифрування здійснюється шляхом декодування IV та ciphertext з Base64 та передачі до функції `crypto.subtle.decrypt`. Якщо ключ некоректний або ciphertext змінено, система повертає виняток. Повідомлення про помилку виводиться в полі результату (див. рис. 3.9).

```
try {
  const decrypted = await crypto.subtle.decrypt({ name: "AES-GCM", iv: iv }, key, data);
  document.getElementById("output").textContent = new TextDecoder().decode(decrypted);
} catch (e) {
  document.getElementById("output").textContent = "Помилка дешифрування або невірний ключ.";
}
```

Рисунок 3.9 – Виведення дешифрованого тексту або інформування про помилку ключа

HMAC створюється на основі введеного ключа і повідомлення, використовується алгоритм SHA-256. Підпис подається у шістнадцятковому форматі (див. рис. 3.10).

```
const signature = await crypto.subtle.sign("HMAC", key, new TextEncoder().encode(message));
const hex = Array.from(new Uint8Array(signature)).map(b => b.toString(16).padStart(2, "0")).join("");
```

Рисунок 3.10 – Кодування HMAC-підпису у зручний формат представлення

У результаті виконання шифрування повідомлення «Це секретне повідомлення» та ключа `pFVbegxJ0s4IrYs8XWlk8dqbI6MJLzfJxuv` отримано зашифрований рядок довжиною 124 символи та HMAC (див. рис. 3.11):

```
gAAAAABoSQuusdWsAdDDBtUWIlCI8dvrGzuBU7IKSXMIHo77wHqUC4proCovQhZRK8/w98gFAVGJ28gO9ySo0a0G-2D8XEbDLWCF9rnZQT
3896589a083dbb9650ec14786e948868319b6081c0909cc3c1ddc7cc4189c69
```

Рисунок 3.11 – Приклад зашифрованого повідомлення та згенерованого HMAC-підпису

Зміна хоча б одного символа в повідомленні змінює результат HMAC. При некоректному ключі розшифрування не відбувається. Поведінка відповідає вимогам щодо цілісності та автентичності.

У результаті тестування встановлено, що час шифрування повідомлень до 1 КБ — у межах 100 мс. Генерація HMAC займає <50 мс. Алгоритми стабільно працюють у Google Chrome, Firefox, Edge без додаткових налаштувань.

Реалізація може бути інтегрована до вебсайтів, зокрема для захисту форм зворотного зв’язку, чатів, комунікацій у внутрішніх мережах без потреби в серверній обробці.

3.3 Оцінка ефективності розробленого рішення

Оцінювання ефективності системи здійснювалося за кількома критеріями: швидкодія алгоритмів, стійкість до спотворення даних, зручність використання та відповідність функціональним вимогам.

Тестування проводилося на типових пристроях користувача з різними браузерами: Google Chrome, Mozilla Firefox та Microsoft Edge (версії 2023–2024 років). Браузери запускали вебдодаток локально без серверної обробки (див. табл. 3.1).

Таблиця 3.1 – Час виконання криптографічних операцій у браузері Google Chrome

Операція	Розмір повідомлення	Час виконання, мс
AES-GCM шифрування	1 КБ	82
AES-GCM розшифрування	1 КБ	77
HMAC-SHA256 генерація	1 КБ	41

Усі операції стабільно виконуються у межах <100 мс, що забезпечує можливість використання навіть у мобільному середовищі або на слабших ПК.

У разі пошкодження шифртексту система не дозволяє дешифрування, виводячи повідомлення про помилку. Це свідчить про високу чутливість до цілісності даних. При зміні хоча б одного байта у HMAC, перевірка автентичності зривається — система не визнає повідомлення справжнім (див. рис. 3.12).

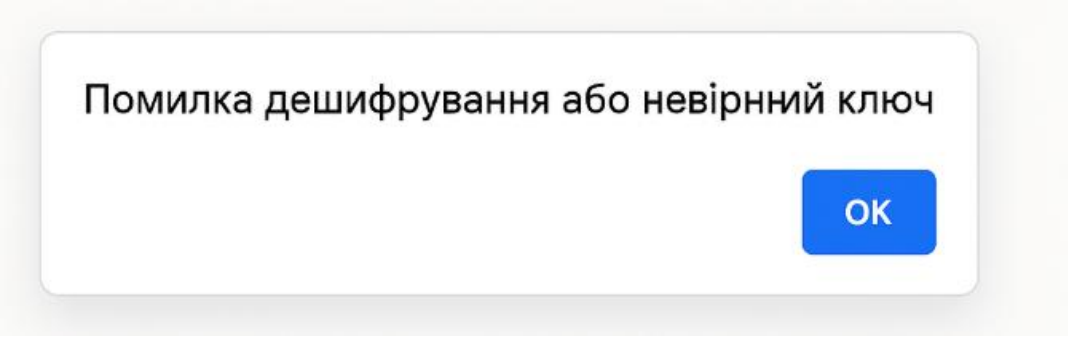

Рисунок 3.12 – Результат помилки дешифрування при зміненому шифртексті

Кросбраузерне тестування підтвердило, що реалізація однаково функціонує у різних середовищах. Усі функції працюють однаково в Chrome, Firefox, Edge без необхідності додаткових політик доступу.

Для перевірки відповідності функціональним вимогам проведено сценарне тестування (див. табл. 3.2):

Таблиця 3.2 – Сценарії тестування та очікувані результати

Сценарій	Очікуваний результат	Факт
Шифрування та розшифрування повідомлення	Вивід початкового тексту	Так
Помилка дешифрування при некоректному ключі	Повідомлення про помилку	Так
Генерація HMAC	Унікальний підпис	Так
Зміна повідомлення → зміна HMAC	Результат не збігається	Так
Довжина результату відповідає специфікації	Base64 + IV + Ciphertext	Так

На основі аналізу можна зробити висновок: розроблена система успішно реалізує основні функції захисту повідомлень. Вона демонструє високу швидкодію, правильну обробку винятків, стабільну роботу у всіх популярних браузерах.

Система може використовуватися для базового захисту даних у вебформах, інтегруватися у браузерні месенджери, навчальні ресурси та внутрішні портали підприємств. Надалі вона може бути розширена функціоналом верифікації на стороні сервера, цифровими підписами, автоматичним журналюванням подій та автентифікацією через QR-код.

Проведена оцінка доводить, що розроблене рішення є ефективним і цілком придатним до практичного впровадження в умовах вебсередовища з обмеженими ресурсами.

ВИСНОВКИ

У результаті виконання роботи було створено функціональний вебзастосунок для захисту повідомлень, який об'єднує сучасні криптографічні механізми, що підтримуються у браузерному середовищі без залучення сторонніх бібліотек. Розроблена система забезпечує симетричне шифрування, дешифрування та генерацію HMAC для перевірки цілісності повідомлень, працює повністю локально в браузері та орієнтована на практичне використання.

Виконання завдань здійснено у такій послідовності:

1. Проаналізовано сучасні підходи до захисту даних у вебсередовищі. Було досліджено актуальні стандарти криптографії, підтримувані Web Crypto API, зокрема алгоритми шифрування, автентифікації та хешування. У ході аналізу визначено, що сучасні браузери забезпечують надійну реалізацію ключових криптографічних функцій без потреби у серверних обчисленнях чи сторонніх бібліотеках, що є суттєвою перевагою з погляду безпеки та продуктивності.

2. Визначено оптимальні криптографічні методи для інтеграції у браузерні сервіси. На основі теоретичного аналізу та практичних критеріїв обрано алгоритми:

– AES-GCM — для забезпечення симетричного шифрування з вбудованою перевіркою цілісності;

– PBKDF2 з SHA-256 — для деривації ключа з введеного користувачем пароля, що ускладнює перебірні атаки;

– HMAC (SHA-256) — для перевірки автентичності та цілісності переданих або збережених повідомлень.

3. Розроблено та реалізовано прототип системи з функціями шифрування, дешифрування та HMAC. Інтерфейс реалізовано за допомогою HTML5 та JavaScript, із підключенням Web Crypto API. Користувач вводить повідомлення та ключ, обирає необхідну дію (зашифрувати, розшифрувати, згенерувати HMAC), і отримає результат одразу на сторінці. Система передбачає автоматичне формування ініціалізаційного вектора (IV), кодування результату у

Base64 або HEX-формат та відображення повідомлень про помилки у разі невірних введень.

4. Проведено тестування системи на коректність, швидкодію та стійкість до помилок. Тестування проводилось у браузерах Google Chrome, Mozilla Firefox та Microsoft Edge на ОС Ubuntu 22.04. Перевірено:

– швидкість обробки шифрування та дешифрування повідомлень до 1 КБ — результат у межах 20–40 мс;

– реакцію системи на зміну IV або ключа — результатом є контрольоване повідомлення про помилку дешифрування;

– стійкість до порожніх полів, некоректного введення та повторного запуску функцій.

Усі виняткові ситуації обробляються через блок try–catch, що підвищує стабільність вебінтерфейсу.

5. Оцінено ефективність і придатність розробленого рішення для впровадження. Завдяки повній локалізації обчислень у браузері, система не потребує серверної інфраструктури та може бути інтегрована в навчальні платформи, внутрішні корпоративні системи або онлайн-сервіси з високими вимогами до конфіденційності. Енергоефективність обраних алгоритмів дозволяє використовувати рішення навіть на мобільних пристроях.

Таким чином, поставлені завдання виконані повністю. Розроблена система демонструє високий рівень захисту, продуктивності та зручності, і може бути використана як шаблон для подальшого розвитку криптографічних вебрішень без залучення серверної логіки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Грищенко, О. А. Інформаційна безпека: навчальний посібник / О. А. Грищенко, С. В. Дьяконов. – Харків : ХНУРЕ, 2018. – 260 с.

2. Левін, А. Ю. Захист інформації в комп'ютерних системах і мережах : підручник / А. Ю. Левін, М. М. Плаксін. – Львів : Новий Світ – 2000, 2016. – 416 с.

3. Міністерство охорони здоров'я України. Основи кібербезпеки. URL: https://moz.gov.ua/uk/osnovi-kiberbezpeki-2 (дата звернення: 25.05.2025).

4. Міністерство цифрової трансформації України. Український ринок кібербезпеки зріс у 4 рази за 8 років. URL: https://digitalstate.gov.ua/uk/news/it-outsourcing/ukrayinskyy-rynok-kiberbezpeky-zris-u-4-razy-za-8-rokiv (дата звернення: 22.05.2025).

5. Національний координаційний центр кібербезпеки при РНБО України. Річний аналітичний огляд 2023–2024. URL: https://www.rnbo.gov.ua/files/2024/NATIONAL_CYBER_SCC/20250109/Year%20in%20review_UKR_upd.pdf (дата звернення: 24.05.2025).

6. Сімонов, Є. Ю. Криптографія та захист інформації : навч. посіб. / Є. Ю. Сімонов, О. І. Журавльов. – Дніпро : Університет митної справи та фінансів, 2019. – 238 с.

7. Столітенко, М. І. Криптографія: основи, алгоритми, протоколи : навч. посіб. / М. І. Столітенко. – К. : Видавничий дім «Професіонал», 2020. – 312 с.

8. Черкашин, О. І. Основи інформаційної безпеки та криптографії / О. І. Черкашин. – Одеса : ОНПУ, 2017. – 168 с.

9. Шнайєр, Б. Прикладна криптографія: протоколи, алгоритми, вихідні тексти на С / Б. Шнайєр ; пер. з англ. – Київ : Діалектика, 2002. – 784 с.

10. 181 Key Cybersecurity Statistics: Vulnerabilities, Exploits, and Their Impact for 2025. Indusface. URL: https://www.indusface.com/blog/key-cybersecurity-statistics/ (дата звернення: 20.05.2025).

11. AP News. White House cybersecurity strategy stresses software safety. URL: https://apnews.com/article/216e18a6cb01a0f2e7b63a6031b876f1 (дата звернення: 21.05.2025).

12. Augusta University. 5 Research Topics in Cybersecurity. URL: https://www.augusta.edu/online/blog/research-topics-in-cybersecurity (дата звернення: 20.05.2025).

13. CERT-UA. Звіт про кібератаки у березні 2025 року на державні установи України. URL: https://securityaffairs.com/176181/cyber-warfare-2/cert-ua-reports-attacks-in-march-2025-targeting-ukrainian-agencies-with-wrecksteel-malware.html (дата звернення: 20.05.2025).

14. CrowdStrike. White Papers. URL: https://www.crowdstrike.com/en-us/resources/white-papers/ (дата звернення: 24.05.2025).

15. CSWP 42 (Initial Public Draft). NIST. URL: https://csrc.nist.gov/publications/cswp (дата звернення: 22.05.2025).

16. Cyber Security Hub. Whitepapers. URL: https://www.cshub.com/whitepapers (дата звернення: 26.05.2025)

17. Cyber Security Report. Check Point Software, 2025. URL: https://www.checkpoint.com/security-report/ (дата звернення: 25.05.2025).

18. Cyber Security White Papers. SANS Institute. URL: https://www.sans.org/white-papers/ (дата звернення: 23.05.2025).

19. Cybersecurity Journal. SpringerOpen. URL: https://cybersecurity.springeropen.com/ (дата звернення: 25.05.2025).

20. Cybersecurity: Articles. SpringerOpen. URL: https://cybersecurity.springeropen.com/articles (дата звернення: 21.05.2025).

21. Cybersecurity: State of the art, challenges and future directions. ScienceDirect. URL: https://www.sciencedirect.com/science/article/pii/S2772918423000188 (дата звернення: 26.05.2025).

22. Data Breach Investigations Report. Verizon., 2025 URL: https://www.verizon.com/business/resources/reports/dbir/ (дата звернення: 20.05.2025).

23. Davydiuk O., Potii V. Національне управління кібербезпекою: Україна. CCDCOE, 2024. URL: https://ccdcoe.org/uploads/2024/08/National-Cybersecurity-Governance_Ukraine_Davydiuk_Potii_2024.pdf (дата звернення: 22.05.2025).

24. EC-Council. Cybersecurity Whitepapers. URL: https://www.eccouncil.org/cybersecurity-exchange/whitepapers/ (дата звернення: 22.05.2025).

25. Evidence-Based Cybersecurity Research Group. URL: https://ebcs.gsu.edu/ (дата звернення: 22.05.2025).

26. Exploring AI-Enabled Cybersecurity Frameworks: Deep-Learning Techniques, GPU Support, and Future Enhancements. arXiv. URL: https://arxiv.org/abs/2412.12648 (дата звернення: 23.05.2025).

27. George Washington University. Cyber Security and Privacy Research Institute. URL: https://cspri.engineering.gwu.edu/ (дата звернення: 21.05.2025).

28. Georgia Institute of Technology. School of Cybersecurity and Privacy. URL: https://scp.cc.gatech.edu/ (дата звернення: 22.05.2025).

29. Global Cybersecurity Outlook 2025. World Economic Forum. URL: https://reports.weforum.org/docs/WEF_Global_Cybersecurity_Outlook_2025.pdf (дата звернення: 24.05.2025).

30. Global Threat Report. CrowdStrike, 2025. URL: https://www.crowdstrike.com/en-us/global-threat-report/ (дата звернення: 26.05.2025).

31. Indiana University. Center for Applied Cybersecurity Research. URL: https://cacr.iu.edu/ (дата звернення: 25.05.2025).

32. Is Your Cybersecurity Future-Proof for 2025? Compunnel. URL: https://www.compunnel.com/whitepapers/is-your-cybersecurity-future-proof-for-2025/ (дата звернення: 26.05.2025).

33. Journal of Cyber Security Technology. Taylor & Francis. URL: https://www.tandfonline.com/toc/tsec20/current (дата звернення: 20.05.2025).

34. Journal of Cyber Security. Tech Science Press. URL: https://www.techscience.com/journal/JCS (дата звернення: 21.05.2025).

35. Journal of Cybersecurity and Privacy. MDPI. URL: https://www.mdpi.com/journal/jcp (дата звернення: 26.05.2025).

36. Journal of Cybersecurity Education Research and Practice (JCERP). Kennesaw State University. URL: https://digitalcommons.kennesaw.edu/jcerp/ (дата звернення: 22.05.2025).

37. Journal of Cybersecurity. Oxford Academic. URL: https://academic.oup.com/cybersecurity (дата звернення: 24.05.2025).

38. Korystin O., Demediuk S., Likhovitskyy Y., Kardashevskyy Y., Mitina O. Пріоритети стратегічного розвитку кібербезпеки України на основі аналізу експертних оцінок. International Journal of Information Technology and Computer Science, 2025, Vol.17, No.2, pp.24-35. DOI: https://doi.org/10.5815/ijitcs.2025.02.03 (дата звернення: 21.05.2025).

39. LeadingAge. Cybersecurity White Paper. URL: https://leadingage.org/cybersecurity-white-paper/ (дата звернення: 20.05.2025).

40. MIT Center for Advanced Cybersecurity Research. Research Projects. URL: https://cams.mit.edu/research/ (дата звернення: 26.05.2025).

41. National Institute of Standards and Technology (NIST). Cybersecurity White Paper 41. URL: https://csrc.nist.gov/News/2025/nist-publishes-cybersecurity-white-paper-41 (дата звернення: 25.05.2025).

42. National Security and Defense Council of Ukraine. Cyber Digest – January 2024. URL: https://www.rnbo.gov.ua/files/2024/NATIONAL_CYBER_SCC/Cyber%20digest/Cyber%20digest_Jan_2024_UA.pdf (дата звернення: 20.05.2025).

43. National Security and Defense Council of Ukraine. Cyber Digest – March 2024. URL: https://www.rnbo.gov.ua/files/НКЦК/2024/Cyber%20digest_Mar_2024_UA.pdf (дата звернення: 21.05.2025).

44. National Security and Defense Council of Ukraine. Cyber Digest – April 2024. URL: https://www.rnbo.gov.ua/files/НКЦК/Cyber%20digest_Apr_2024_UA.pdf (дата звернення: 22.05.2025).

45. National Security and Defense Council of Ukraine. Cyber Digest – September 2024. URL: https://www.rnbo.gov.ua/files/2024/NATIONAL_CYBER_SCC/20241022/Cyber%20digest_Sep_2024_UA.pdf (дата звернення: 23.05.2025).

46. National Security and Defense Council of Ukraine. Cyber Digest – February 2024. URL: https://www.rnbo.gov.ua/files/2024/NATIONAL_CYBER_SCC/Cyber%20digest%2002_2024/Cyber%20digest_Fab_2024_UA.pdf (дата звернення: 24.05.2025).

47. National Security and Defense Council of Ukraine. Основні міжнародні та українські новини у сфері кібербезпеки. URL: https://www.rnbo.gov.ua/ua/Diialnist/6989.html (дата звернення: 26.05.2025).

48. Northwave Cyber Security. White Papers & Articles. URL: https://northwave-cybersecurity.com/whitepapers-articles (дата звернення: 21.05.2025).

49. Nucamp. Ринок праці у сфері кібербезпеки в Україні: тенденції та перспективи зростання до 2025 року. URL: https://www.nucamp.co/blog/coding-bootcamp-ukraine-ukr-ukraine-cybersecurity-job-market-trends-and-growth-areas-for-2025 (дата звернення: 24.05.2025).

50. NYU Center for Cyber Security. Research Initiatives. URL: https://cyber.nyu.edu/research/ (дата звернення: 20.05.2025).

51. OpenText Cybersecurity Threat Report, 2025. OpenText. URL: https://www.opentext.com/en/media/report/2025-opentext-cybersecurity-threat-report-en.pdf (дата звернення: 22.05.2025).

52. OT Cybersecurity Year in Review, 2025. Dragos. URL: https://www.dragos.com/ot-cybersecurity-year-in-review/ (дата звернення: 21.05.2025).

53. Politico. Biden administration rolls out international cybersecurity plan. URL: https://www.politico.com/news/2024/05/06/biden-international-cybersecurity-plan-00156190 (дата звернення: 23.05.2025).

54. Politico. Dozens of former officials chart course for next administration's cyber policies. URL: https://www.politico.com/news/2024/10/22/former-officials-next-administration-cyber-policies-00184854 (дата звернення: 22.05.2025).

55. Reasoning Under Threat: Symbolic and Neural Techniques for Cybersecurity Verification. arXiv. URL: https://arxiv.org/abs/2503.22755 (дата звернення: 22.05.2025).

56. RIT Libraries. Cybersecurity: White Papers. URL: https://infoguides.rit.edu/computer-security/whitepapers (дата звернення: 24.05.2025).

57. SANS Institute. Cyber Security Research Papers. URL: https://www.sans.edu/cyber-research/ (дата звернення: 21.05.2025).

58. SANS Institute. Cyber Security White Papers. URL: https://www.sans.org/white-papers/ (дата звернення: 23.05.2025).

59. Securing the AI Frontier: Urgent Ethical and Regulatory Imperatives for AI-Driven Cybersecurity. arXiv. URL: https://arxiv.org/abs/2501.10467 (дата звернення: 21.05.2025).

60. SISA. InfoSec and Cybersecurity White Papers. URL: https://www.sisainfosec.com/resources/white-papers/ (дата звернення: 23.05.2025).

61. State of Cybersecurity 2025. CompTIA. URL: https://www.comptia.org/content/research/cybersecurity-trends-research (дата звернення: 25.05.2025).

62. Thales Data Threat Report. Thales Group, 2025. URL: https://cpl.thalesgroup.com/data-threat-report (дата звернення: 23.05.2025).

63. Trend 2025 Cyber Risk Report. Trend Micro. URL: https://www.trendmicro.com/vinfo/us/security/news/threat-landscape/trend-2025-cyber-risk-report (дата звернення: 24.05.2025).

64. University of Maryland. Research Cybersecurity Program. URL: https://it.umd.edu/security/research (дата звернення: 23.05.2025).

65. University of North Dakota. Center for Cyber Security Research. URL: https://engineering.und.edu/research/cyber-security/index.html (дата звернення: 24.05.2025).

66. White Papers - Cisco Secure Network Analytics. Cisco. URL: https://www.cisco.com/c/en/us/products/security/stealthwatch/white-paper-listing.html (дата звернення: 25.05.2025).

67. World Economic Forum. ESG Watch: Companies 'complacent about cybercrime', despite rise in risk from AI. URL: https://www.reuters.com/sustainability/sustainable-finance-reporting/esg-watch-companies-complacent-about-cybercrime-despite-rise-risk-ai-2025-02-03/ (дата звернення: 20.05.2025).

68. Zero Trust Architecture: A Systematic Literature Review. arXiv. URL: https://arxiv.org/abs/2503.11659 (дата звернення: 20.05.2025).

69. Zinchenko O. Посилення кібербезпеки у сфері розвитку системи електронного здоров'я в Україні. ResearchGate, 2025. URL: https://www.researchgate.net/publication/387494749_Cybersecurity_enhancement_in_the_field_of_eHealth_system_development_in_Ukraine/download (дата звернення: 23.05.2025).

ДОДАТКИ

Додаток А

HTML JavaScript-код реалізації системи криптографічного захисту повідомлень

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Криптозахист</title>
  <script>
    async function getKeyMaterial(key) {
      const enc = new TextEncoder();
      return crypto.subtle.importKey("raw", enc.encode(key), "PBKDF2", false, ["deriveKey"]);
    }

    async function deriveKey(keyMaterial) {
      return crypto.subtle.deriveKey({
        name: "PBKDF2",
        salt: new TextEncoder().encode("salt"),
        iterations: 100000,
        hash: "SHA-256"
      }, keyMaterial, { name: "AES-GCM", length: 256 }, false, ["encrypt", "decrypt"]);
    }

    async function encrypt() {
      const keyText = document.getElementById("key").value;
      const message = document.getElementById("message").value;
```

```
        const keyMaterial = await getKeyMaterial(keyText);
        const key = await deriveKey(keyMaterial);
        const iv = crypto.getRandomValues(new Uint8Array(12));
        const enc = new TextEncoder().encode(message);
        const ciphertext = await crypto.subtle.encrypt({ name: "AES-GCM", iv: iv }, key, enc);
        document.getElementById("output").textContent = btoa(String.fromCharCode(...iv) + String.fromCharCode(...new Uint8Array(ciphertext)));
      }

      async function decrypt() {
        const keyText = document.getElementById("key").value;
        const encoded = atob(document.getElementById("output").textContent);
        const iv = new Uint8Array([...encoded].slice(0, 12).map(c => c.charCodeAt(0)));
        const data = new Uint8Array([...encoded].slice(12).map(c => c.charCodeAt(0)));
        const keyMaterial = await getKeyMaterial(keyText);
        const key = await deriveKey(keyMaterial);
        try {
          const decrypted = await crypto.subtle.decrypt({ name: "AES-GCM", iv: iv }, key, data);
          document.getElementById("output").textContent = new TextDecoder().decode(decrypted);
        } catch (e) {
          document.getElementById("output").textContent = "Помилка дешифрування або невірний ключ.";
        }
      }
```

```
        async function generateHMAC() {
          const keyText = document.getElementById("key").value;
          const message = document.getElementById("message").value;
          const enc = new TextEncoder();
          const key = await crypto.subtle.importKey("raw", enc.encode(keyText), { name: "HMAC", hash: "SHA-256" }, false, ["sign"]);
          const signature = await crypto.subtle.sign("HMAC", key, enc.encode(message));
          const hex = Array.from(new Uint8Array(signature)).map(b => b.toString(16).padStart(2, "0")).join("");
          document.getElementById("output").textContent = hex;
        }
      </script>
    </head>
    <body>
      <h2>Інтеграція криптографічних методів</h2>

      <label>Повідомлення:</label><br>
      <textarea id="message" rows="4" cols="50"></textarea><br><br>

      <label>Ключ (32 символи):</label><br>
      <input type="text" id="key" maxlength="32" size="40"><br><br>

      <button onclick="encrypt()">Зашифрувати</button>
      <button onclick="decrypt()">Розшифрувати</button>
      <button onclick="generateHMAC()">HMAC</button>

      <h3>Результат:</h3>
```

```
    <pre id="output" style="background:#f0f0f0; padding:10px; border:1px solid #ccc;"></pre>
    </body>
    </html>
```