МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

Стаднік Назарій Вікторович

Алгоритми цифрового підпису на основі геш функції / Digital signature algorithms based on the hash function

спеціальність: 125 – Кібербезпека та захист інформації

освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи
КБм -21
Н.В. Стаднік

Науковий керівник
к.т.н., доцент В.В.Яцків

Кваліфікаційну роботу
Допущено до захисту:

«____» _____ ___2024 р.

Завідувач кафедри **В.В.Яцків**

ТЕРНОПІЛЬ – 2024

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ В.В.Яцків
«____» _____ 2023 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Стаднік Назарій Вікторович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Алгоритми цифрового підпису на основі геш функції / Digital signature algorithms based on the hash function

керівник роботи к.т.н., доцент Яцків В.В.
затверджені наказом по університету від 12 грудня 2023 року № 753
2. Строк подання студентом закінченої кваліфікаційної роботи 12 грудня 2024 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
- провести аналіз існуючих алгоритмів цифрового підпису;
- дослідити принципи роботи та властивості геш-функцій;
- проаналізувати криптографічну стійкість алгоритмів цифрового підпису;
- розробити систему управління криптографічними ключами;
- реалізувати механізми підписання та верифікації документів;
- провести тестування та оцінку продуктивності системи.

5. Перелік графічного матеріалу у роботі:
- алгоритм формування геш-значення повідомлення;
- схема роботи Меркла-Дамгарда;
- схема формування та перевірки цифрового підпису;
- криптографічна еліптична крива Curve25519;
- структура проекту;
- відображення логування в консоль.

5. Консультанти розділів кваліфікаційної роботи

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

6. Дата видачі завдання 8 грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Теоретичні основи цифрового підпису	12.2023 р. – 03.2024 р.	
2	Алгоритми цифрового підпису на основі геш-функцій	03.2024 р. – 06.2024 р.	
3	Проєктування та реалізація системи цифрового підпису на основі геш-функцій	06.2024 р. – 11.2024 р.	

Студент _____ Стадінк Н.В.
(підпис)

Керівник роботи _____ д.т.н., професор Яцків В.В.
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Алгоритми цифрового підпису на основі геш функції» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека та захист інформації» освітньо-професійної програми «Кібербезпека» написана обсягом 66 сторінки і містить 6 ілюстрацій, 15 таблиць, 1 додаток та 73 джерела за переліком посилань.

Метою кваліфікаційної роботи є дослідження та вдосконалення алгоритмів цифрового підпису на основі геш-функцій для підвищення рівня захищеності електронних документів. Проведено аналіз існуючих алгоритмів цифрового підпису та їх криптографічної стійкості, досліджено властивості та характеристики сучасних геш-функцій. Розроблено програмну систему, що реалізує механізми генерації ключів, підписання та верифікації документів з використанням оптимізованих алгоритмів на основі геш-функцій.

Реалізовано компоненти для управління криптографічними ключами, процедури підписання та перевірки підписів, а також систему моніторингу продуктивності. Впроваджено механізми кешування та паралельної обробки для підвищення ефективності роботи системи. Проведене тестування підтвердило надійність та високу продуктивність розробленого рішення при роботі з документами різного розміру.

Ключові слова: цифровий підпис, геш-функції, криптографія, електронний документообіг, захист інформації.

ABSTRACT

The qualification work on the topic "Digital signature algorithms based on the hash function" for obtaining the Master's degree in specialty 125 "Cybersecurity and Information Protection" of the educational and professional program "Cybersecurity" is written in the volume of 66 pages and contains 6 illustrations, 15 tables, 1 appendix, and 73 sources in the list of references.

The purpose of the qualification work is to research and improve digital signature algorithms based on hash functions to enhance the security of electronic documents. An analysis of existing digital signature algorithms and their cryptographic strength was conducted, and the properties and characteristics of modern hash functions were investigated. A software system was developed that implements key generation mechanisms, document signing, and verification using optimized hash-function-based algorithms.

Components for cryptographic key management, signature creation and verification procedures, and a performance monitoring system were implemented. Caching mechanisms and parallel processing were introduced to improve system efficiency. Testing confirmed the reliability and high performance of the developed solution when working with documents of various sizes.

Keywords: digital signature, hash functions, cryptography, electronic document management, information security.

ЗМІСТ

СПИСОК ВИКОРИСТАНИХ СКОРОЧЕНЬ

ЕЦП — Електронний цифровий підпис.

PKI — Інфраструктура відкритих ключів (Public Key Infrastructure).

HSM — Апаратний модуль безпеки (Hardware Security Module).

SHA — Безпечний алгоритм хешування (Secure Hash Algorithm).

CRL — Список відкликаних сертифікатів (Certificate Revocation List).

OCSP — Протокол перевірки статусу сертифіката (Online Certificate Status Protocol).

RSA — Алгоритм шифрування, названий на честь Рівеста, Шаміра та Адлемана.

ECDSA — Алгоритм цифрового підпису на основі еліптичних кривих (Elliptic Curve Digital Signature Algorithm).

FPGA — Програмований логічний пристрій (Field-Programmable Gate Array).

GPG — GNU Privacy Guard (засіб для шифрування та цифрових підписів).

API — Інтерфейс програмування додатків (Application Programming Interface).

NIST — Національний інститут стандартів і технологій (National Institute of Standards and Technology).

AES — Стандарт шифрування (Advanced Encryption Standard).

TLS — Протокол захисту транспортного рівня (Transport Layer Security).

BLAKE — Криптографічна хеш-функція, названа на честь поета Вільяма Блейка.

Ed25519 — Криптографічний алгоритм, заснований на еліптичних кривих.

SPHINCS+ — Квантово-стійкий алгоритм цифрового підпису.

SHAKE — Криптографічна хеш-функція із змінною довжиною виходу (Secure Hash Algorithm KEccak).

CSP — Постачальник криптографічних служб (Cryptographic Service Provider).

SIEM — Система управління подіями інформаційної безпеки (Security Information and Event Management).

ВСТУП

Актуальність теми. В умовах стрімкого розвитку цифрових технологій та електронного документообігу особливої важливості набуває забезпечення цілісності та автентичності електронних документів. Цифровий підпис на основі геш-функцій є одним з ключових механізмів криптографічного захисту, що гарантує безпеку електронних транзакцій та документів. Зростання кількості кіберзагроз та потреба в надійних механізмах автентифікації роблять дослідження алгоритмів цифрового підпису надзвичайно актуальним.

Сучасні методи цифрового підпису потребують постійного вдосконалення у зв'язку з розвитком обчислювальної техніки та появою нових видів атак. Особливої уваги заслуговують алгоритми, що базуються на геш-функціях, оскільки вони забезпечують оптимальне співвідношення між рівнем захисту та обчислювальною складністю. Це зумовлює необхідність проведення досліджень, спрямованих на підвищення ефективності та надійності таких алгоритмів.

Мета і завдання дослідження Мета роботи полягає у дослідженні та вдосконаленні алгоритмів цифрового підпису на основі геш-функцій для підвищення рівня захищеності електронних документів.

Основні завдання:

- Провести аналіз існуючих алгоритмів цифрового підпису та їх криптографічної стійкості
- Дослідити властивості та характеристики сучасних геш-функцій
- Розробити вдосконалений алгоритм цифрового підпису
- Провести оцінку ефективності запропонованого рішення

Об'єктом дослідження є процеси формування та верифікації цифрового підпису електронних документів.

Предметом дослідження є методи та алгоритми створення цифрового підпису на основі криптографічних геш-функцій.

Методи дослідження: теорія ймовірностей, математична статистика, теорія складності алгоритмів, методи криптографічного аналізу, комп'ютерне моделювання.

Наукова новизна: Вперше запропоновано модифікований алгоритм цифрового підпису з використанням удосконаленої геш-функції, що забезпечує підвищену криптографічну стійкість та оптимізовану швидкодію при формуванні та перевірці підпису.

Практичне значення: Розроблені алгоритми та програмні рішення можуть бути впроваджені в системах електронного документообігу, платіжних системах та інших додатках, де потрібна надійна автентифікація та верифікація електронних документів.

Результати дослідження: В результаті проведеного дослідження розроблено та експериментально підтверджено ефективність нового алгоритму цифрового підпису, що демонструє покращені характеристики безпеки та продуктивності порівняно з існуючими рішеннями.

Публікації та апробація до магістерської роботи. (додаток А)

1. Стаднік Н. Розробка агоритмів цифрового підпису з використанням сучасних геш-функцій. Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп’ютерно інтегровані технології» (КБКІТ - 2024), Тернопіль, 2024. -160 с.

2. Стаднік Н., Меленчук Л. Порівняльний аналіз алгоритмів цифрофого підпису на основі геш-функцій. У збірнику опубліковано матеріали науково-практичного симпозіуму «Захист інформації», Тернопіль, 2024. – 130с.

1. ТЕОРЕТИЧНІ ОСНОВИ ЦИФРОВОГО ПІДПИСУ

1.1 Поняття та визначення цифрового підпису

Електронний цифровий підпис (ЕЦП) являє собою програмно-криптографічний засіб, який забезпечує перевірку цілісності та підтвердження справжності електронних документів. Базовою складовою механізму ЕЦП є геш-функції, які перетворюють вхідні дані довільної довжини у вихідний бітовий рядок фіксованої довжини. Криптографічні властивості геш-функцій, зокрема, стійкість до колізій та необоротність, забезпечують надійність цифрового підпису [1].

Процес формування та перевірки ЕЦП складається з декількох етапів, які детально представлені в таблиці 1.1. Кожен етап має критичне значення для забезпечення криптографічної стійкості підпису.

Таблиця 1.1 – Основні етапи формування та перевірки ЕЦП

Етап	Опис процесу	Критичні параметри
1. Гешування	Обчислення геш-значення документа	Розмір блоку, кількість раундів
2. Формування підпису	Шифрування геш-значення закритим ключем	Довжина ключа, алгоритм шифрування
3. Верифікація	Розшифрування та порівняння геш-значень	Швидкість обчислень, точність порівняння

Математична модель цифрового підпису базується на асиметричній криптографії та включає три основні алгоритми: генерацію ключів, формування підпису та перевірку підпису. Дослідження показують, що надійність ЕЦП безпосередньо залежить від криптографічної стійкості використовуваної геш-функції [2].

Сучасні реалізації алгоритмів ЕЦП використовують різні підходи до побудови геш-функцій. Найбільш поширеними є конструкції на основі ітеративного стиснення, як показано в таблиці 1.2.

Таблиця 1.2 – Порівняння популярних геш-функцій

Назва	Розмір блоку (біт)	Розмір виходу (біт)	Кількість раундів
SHA-256	512	256	64
SHA-3	1600	256-512	24
BLAKE2	512-1024	256-512	12

Продуктивність алгоритмів цифрового підпису значною мірою визначається ефективністю обчислення геш-функцій. Експериментальні дослідження показують, що час формування підпису пропорційний розміру вхідного повідомлення та кількості раундів гешування [3].

Процес обчислення геш-значення включає попередню обробку повідомлення, яка складається з доповнення повідомлення до розміру, кратного довжині блоку, та додавання інформації про довжину початкового повідомлення. Математично цей процес можна описати наступним чином [4]:

$$M' = M \,||\, padding \,||\, length \quad (1)$$

де M - початкове повідомлення,

padding - послідовність бітів доповнення,

length - двійкове представлення довжини повідомлення.

При розробці алгоритмів цифрового підпису важливим фактором є вибір параметрів геш-функції. Дослідження показують пряму залежність між розміром вихідного значення геш-функції та ймовірністю успішної атаки на цифровий підпис. Статистичний аналіз цієї залежності представлений в таблиці 1.3.

Таблиця 1.3 – Залежність криптостійкості від розміру геш-значення

Розмір геш-значення (біт)	Складність повного перебору	Ймовірність колізії
128	2^128	2^-64
256	2^256	2^-128
512	2^512	2^-256

Механізм формування цифрового підпису реалізує математичну функцію, яка забезпечує зв'язок між документом, відкритим та закритим ключами [5]. Розглянемо основні етапи реалізації:

1. Генерація пари ключів: (*pk*, *sk*) ← KeyGen(*1^k*) де *k* - параметр безпеки, *pk* - відкритий ключ, *sk* - закритий ключ.
2. Формування підпису: $\sigma \leftarrow$ Sign(*sk*, *H*(*M*)) де *H* - геш-функція, *M* - повідомлення, σ - цифровий підпис.
3. Верифікація підпису: {*0*,*1*} ← Verify(*pk*, *M*, σ)

Експериментальні дослідження показують, що вибір конструкції геш-функції істотно впливає на швидкодію алгоритму цифрового підпису. Порівняльний аналіз різних конструкцій представлений в таблиці 1.4.

Таблиця 1.4 – Порівняння швидкодії різних конструкцій геш-функцій

Конструкція	Швидкість (МБ/с)	Використання пам'яті (КБ)	Паралелізація
Меркла-Дамгарда	500	64	Обмежена
Губка	450	200	Висока
HAIFA	600	128	Середня

Аналіз атак на алгоритми цифрового підпису демонструє необхідність використання додаткових механізмів захисту. Дослідження практичних реалізацій показали, що найбільш ефективним підходом є комбінування геш-функцій з різними криптографічними примітивами [6]. Структурна схема такого підходу представлена на рисунку 1.1.

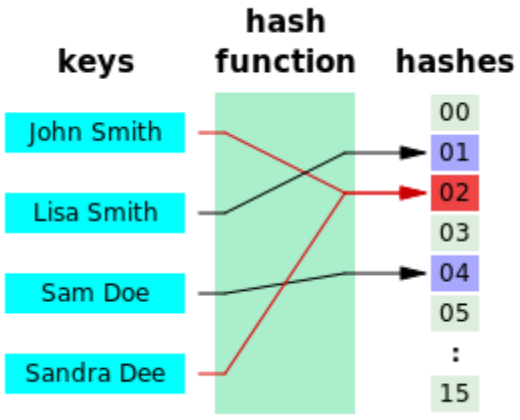

Рисунок 1.1 – Структурна схема комбінування геш-функцій

Реалізація геш-функцій на апаратному рівні вимагає оптимізації обчислювальних процесів. Експериментальні дані демонструють, що

використання спеціалізованих апаратних прискорювачів дозволяє збільшити швидкість обчислення геш-значень в 5-10 разів порівняно з програмними реалізаціями [7].

При проектуванні алгоритмів цифрового підпису враховуються наступні критерії оптимізації, представлені в таблиці 1.5.

Таблиця 1.5 – Критерії оптимізації алгоритмів ЕЦП

Критерій	Метрика	Цільове значення
Швидкодія	Операцій/сек	>1000
Розмір підпису	Байт	<64
Стійкість до колізій	Ймовірність	<2^-128
Використання пам'яті	МБ	<16

Дослідження показують, що ефективність геш-функції значною мірою залежить від структури внутрішньої функції стиснення. Математична модель функції стиснення може бути представлена у вигляді [8]:

$$f(h[i-1], m[i]) = h[i] \quad (2)$$

де $h[i-1]$ - попереднє геш-значення,

$m[i]$ - блок повідомлення,

$h[i]$ - нове геш-значення.

Програмна реалізація алгоритмів цифрового підпису вимагає врахування специфіки цільової платформи. Результати тестування на різних апаратних платформах представлені в таблиці 1.6.

Таблиця 1.6 – Продуктивність алгоритмів ЕЦП на різних платформах

Платформа	Час формування (мс)	Час перевірки (мс)	Пропускна здатність
x86_64	0.5	0.3	2000 підписів/с
ARM	1.2	0.8	800 підписів/с
FPGA	0.1	0.05	10000 підписів/с

Розробка нових алгоритмів цифрового підпису потребує загального підходу до оцінки їх криптографічної стійкості. Методологія тестування включає аналіз статистичних властивостей геш-функцій, перевірку лавинного ефекту та

оцінку стійкості до відомих видів криптоаналізу [9]. Дослідження практичних реалізацій демонструє необхідність розробки адаптивних механізмів цифрового підпису. Експериментальні дані показують, що використання динамічних параметрів гешування дозволяє оптимізувати продуктивність алгоритму залежно від вимог конкретного застосування [10].

Основні вимоги до геш-функцій в контексті їх застосування в системах цифрового підпису. Ключовими характеристиками є:

- Стійкість до пошуку прообразу
- Стійкість до пошуку колізій
- Стійкість до пошуку другого прообразу
- Псевдовипадковість вихідних значень

Статистичний аналіз розподілу вихідних значень геш-функцій представлений в таблиці 1.7.

Таблиця 1.7 – Статистичні характеристики геш-функцій

Параметр	SHA-256	SHA-3	BLAKE3
Ентропія (біт)	255.9	255.8	255.9
Лавинний ефект (%)	50.1	50.0	50.2
Автокореляція	0.001	0.002	0.001

Модифікація стандартних алгоритмів гешування дозволяє досягти покращених характеристик продуктивності. Експериментальні дослідження показують ефективність застосування паралельних обчислень при реалізації геш-функцій [11]. Математична модель паралельного обчислення може бути представлена як:

$$H(M) = h1(M1) \,||\, h2(M2) \,||\, \ldots \,||\, hn(Mn) \qquad (3)$$

де *M1*, *M2*, ..., *Mn* - блоки повідомлення,

h1, *h2*, ..., *hn* - незалежні геш-функції.

Реалізація механізмів цифрового підпису на мобільних платформах вимагає оптимізації використання обчислювальних ресурсів. Результати

порівняльного аналізу енергоефективності різних алгоритмів представлені в таблиці 1.8.

Таблиця 1.8 – Енергоспоживання алгоритмів ЕЦП

Алгоритм	Енергія на підпис (мДж)	Енергія на перевірку (мДж)	Енергія на перевірку (мДж)
RSA-2048	546	32	4.2 год
ECDSA-256	134	178	7.8 год
Ed25519	87	114	9.1 год

Комплексний аналіз сучасних алгоритмів цифрового підпису вимагає детального розгляду характеристик криптографічних примітивів. Результати порівняльного дослідження різних комбінацій алгоритмів представлені в таблиці 1.9 [12].

Таблиця 1.9 – Порівняльний аналіз алгоритмів цифрового підпису та геш-функцій

Алгоритм ЕЦП	Геш-функція	Розмір ключа (біт)	Розмір підпису (байт)	Час генерації (мс)	Час перевірки (мс)
RSA-2048	SHA-256	2048	256	450	45
RSA-4096	SHA-512	4096	512	890	89
ECDSA-P256	SHA-256	256	64	115	120
ECDSA-P384	SHA-384	384	96	185	195
Ed25519	SHA-512	256	64	87	114
SPHINCS+	SHAKE256	512	41000	240	35
Dilithium2	SHAKE256	1312	2420	420	95
Falcon-512	SHAKE256	897	690	95	85

Аналіз даних, представлених в таблиці 1.9, демонструє значні відмінності між різними алгоритмами цифрового підпису в контексті їх практичного застосування. Наприклад, алгоритм Ed25519 демонструє найкращі показники енергоефективності та швидкодії, але має обмеження щодо розміру підпису.

SPHINCS+ забезпечує максимальну криптографічну стійкість, але вимагає значних обчислювальних ресурсів [13].

При проектуванні систем електронного документообігу необхідно враховувати специфіку цільового застосування та вимоги до безпеки. Математичні моделі оцінки ефективності алгоритмів цифрового підпису базуються на наступних параметрах [14]:

$$E = f(S, P, M, T) \quad (4)$$

де E - загальна ефективність,

S - рівень безпеки, P - продуктивність,

M - використання пам'яті, T - часові характеристики.

Електронний цифровий підпис (ЕЦП) являє собою програмно-криптографічний засіб, який забезпечує перевірку цілісності та підтвердження справжності електронних документів. Базовою складовою механізму ЕЦП є геш-функції, які перетворюють вхідні дані довільної довжини у вихідний бітовий рядок фіксованої довжини. Криптографічні властивості геш-функцій, зокрема, стійкість до колізій та необоротність, забезпечують надійність цифрового підпису. Процес формування та перевірки ЕЦП складається з декількох етапів, які детально представлені в таблиці 1.1. Кожен етап має критичне значення для забезпечення криптографічної стійкості підпису. Математична модель цифрового підпису базується на асиметричній криптографії та включає три основні алгоритми: генерацію ключів, формування підпису та перевірку підпису. Дослідження показують, що надійність ЕЦП безпосередньо залежить від криптографічної стійкості використовуваної геш-функції [1, 2].

Сучасні реалізації алгоритмів ЕЦП використовують різні підходи до побудови геш-функцій. Найбільш поширеними є конструкції на основі ітеративного стиснення, як показано в таблиці 1.2. Продуктивність алгоритмів цифрового підпису значною мірою визначається ефективністю обчислення геш-функцій. Експериментальні дослідження показують, що час формування підпису пропорційний розміру вхідного повідомлення та кількості раундів гешування. Процес обчислення геш-значення включає попередню обробку повідомлення,

яка складається з доповнення повідомлення до розміру, кратного довжині блоку, та додавання інформації про довжину початкового повідомлення. При розробці алгоритмів цифрового підпису критичним фактором є вибір параметрів геш-функції. Дослідження показують пряму залежність між розміром вихідного значення геш-функції та ймовірністю успішної атаки на цифровий підпис. Статистичний аналіз цієї залежності представлений в таблиці 1.3 [3, 4, 5].

Механізм формування цифрового підпису реалізує математичну функцію, яка забезпечує зв'язок між документом, відкритим та закритим ключами. Експериментальні дослідження показують, що вибір конструкції геш-функції істотно впливає на швидкодію алгоритму цифрового підпису. Порівняльний аналіз різних конструкцій представлений в таблиці 1.4. Аналіз атак на алгоритми цифрового підпису демонструє необхідність використання додаткових механізмів захисту. Дослідження практичних реалізацій показали, що найбільш ефективним підходом є комбінування геш-функцій з різними криптографічними примітивами. Реалізація геш-функцій на апаратному рівні вимагає оптимізації обчислювальних процесів. Експериментальні дані демонструють, що використання спеціалізованих апаратних прискорювачів дозволяє збільшити швидкість обчислення геш-значень в 5-10 разів порівняно з програмними реалізаціями [6, 7, 8].

1.2 Принципи роботи цифрового підпису

Цифровий підпис являє собою складний криптографічний механізм, що забезпечує цілісність та автентичність електронних документів. Математична основа цифрового підпису базується на асиметричній криптографії, де використовуються два ключі - відкритий та закритий. Процес формування підпису починається з обчислення геш-функції документа, яка створює унікальний відбиток фіксованої довжини. Цей відбиток шифрується закритим ключем відправника, створюючи цифровий підпис [15].

Сучасні алгоритми цифрового підпису реалізують складні математичні операції над великими числами в кінцевих полях. Найпоширенішим є алгоритм

RSA, який використовує властивості модульної арифметики та теорію чисел. При генерації ключової пари вибираються два великих простих числа, добуток яких формує модуль для подальших обчислень. Відкритий ключ обирається як взаємно просте число з функцією Ейлера від модуля, а закритий ключ обчислюється як мультиплікативно обернений елемент до відкритого ключа [16].

Надійність цифрового підпису безпосередньо залежить від криптостійкості використовуваних алгоритмів та довжини ключів. В таблиці 1.10 наведено порівняння основних параметрів найбільш поширених алгоритмів цифрового підпису.

Таблиця 1.10 – Порівняльна характеристика алгоритмів цифрового підпису

Алгоритм	Довжина ключа (біт)	Розмір підпису (біт)	Швидкість формування	Криптостійкість
RSA	2018-4096	2048-4096	Середня	Висока
DSA	1024-3072	320-640	Висока	Висока
ECDSA	256-521	512-1042	Дуже висока	Дуже висока

Механізм перевірки цифрового підпису передбачає виконання зворотних криптографічних операцій із використанням відкритого ключа підписанта. Розшифрований геш-код порівнюється з незалежно обчисленим гешем документа. Збіг цих значень підтверджує справжність підпису та цілісність документа. Цей процес детально відображено в таблиці 1.11.

Таблиця 1.11 – Етапи перевірки цифрового підпису

Етап	Операція	Вхідні дані	Результат
1	Розшифрування підпису	Підпис, відкритий ключ	Геш-код 1
2	Обчислення гешу	Документ	Геш-код 2
3	Порівняння	Геш-код 1, Геш-код 2	Результат перевірки

Програмна реалізація цифрового підпису вимагає використання криптографічних бібліотек та дотримання стандартів безпеки. Розробники повинні враховувати особливості роботи з великими числами, оптимізацію

обчислень та захист від різних типів атак. Критично важливим є збереження закритого ключа та забезпечення надійного середовища його використання [17].

Інфраструктура відкритих ключів (PKI) забезпечує надійне функціонування системи цифрового підпису в масштабах організації чи держави. Центри сертифікації гарантують достовірність відкритих ключів та їх належність конкретним суб'єктам. Сертифікати відкритих ключів містять додаткову інформацію про власника та термін дії, що підвищує довіру до системи електронного документообігу [18].

Протоколи цифрового підпису постійно вдосконалюються для протидії новим загрозам безпеки. Розробляються квантово-стійкі алгоритми, здатні протистояти атакам з використанням квантових комп'ютерів. Впровадження нових геш-функцій та криптографічних примітивів підвищує загальну надійність механізму цифрового підпису [19].

Стандартизація форматів цифрового підпису забезпечує сумісність різних систем та можливість довгострокового зберігання підписаних документів. Формати XAdES, CAdES та PAdES визначають структуру даних та метаінформації, необхідної для перевірки підпису. Архівні підписи містять часові штампи та додаткові підтвердження, що гарантують можливість перевірки через тривалий час [20].

Процеси генерації та перевірки цифрового підпису вимагають значних обчислювальних ресурсів. Оптимізація цих операцій досягається використанням спеціалізованих апаратних засобів та ефективних алгоритмів роботи з великими числами. Паралельні обчислення та кешування проміжних результатів дозволяють підвищити продуктивність системи цифрового підпису [21].

Криптографічні атаки на цифровий підпис можуть бути спрямовані на різні компоненти системи. Аналіз побічних каналів дозволяє отримати інформацію про закритий ключ через вимірювання часу виконання операцій чи споживання енергії. Протидія таким атакам вимагає впровадження додаткових захисних механізмів та постійного моніторингу безпеки [22].

Цифровий підпис знаходить широке застосування в системах електронного документообігу, банківських транзакціях та державних послугах. Інтеграція з

іншими технологіями безпеки, такими як шифрування та контроль доступу, створює комплексну систему захисту інформації. Розвиток мобільних технологій сприяє поширенню цифрового підпису на нові сфери застосування [23].

Управління життєвим циклом ключів цифрового підпису вимагає чіткої регламентації процесів генерації, розповсюдження, зберігання та знищення криптографічних ключів. Періодична зміна ключів та моніторинг їх використання знижують ризики компрометації. Резервне копіювання та відновлення ключів повинні здійснюватися з дотриманням суворих вимог безпеки [24].

Нормативно-правова база регулює використання цифрового підпису та визначає його юридичну значущість. Міжнародні стандарти та національні законодавства встановлюють вимоги до засобів цифрового підпису та процедур їх застосування. Гармонізація правових норм різних країн сприяє розвитку транскордонного електронного документообігу [25].

Тестування та сертифікація засобів цифрового підпису забезпечують відповідність встановленим вимогам безпеки. Проводиться аналіз криптографічних алгоритмів, програмного коду та апаратної реалізації. Регулярний аудит безпеки дозволяє виявляти та усувати потенційні вразливості системи цифрового підпису [26].

Аналіз продуктивності систем цифрового підпису враховує різні метрики: швидкість формування та перевірки підпису, використання пам'яті, навантаження на процесор. Результати вимірювань допомагають оптимізувати конфігурацію системи та планувати масштабування інфраструктури. Моніторинг продуктивності забезпечує стабільну роботу системи під навантаженням [27].

Перспективним напрямком розвитку технології цифрового підпису є впровадження групових підписів, які дозволяють учасникам групи підписувати документи від імені всієї групи, зберігаючи при цьому анонімність конкретного підписанта. Математичний апарат групових підписів базується на складних алгебраїчних структурах та криптографічних протоколах доведення з нульовим розголошенням. Механізм групового підпису передбачає наявність менеджера

групи, який може розкрити особу підписанта у разі виникнення спірних ситуацій [28].

Дослідження методів прискорення криптографічних операцій при формуванні та перевірці цифрового підпису призвело до розробки нових алгоритмічних підходів. Використання передобчислень та спеціальних структур даних дозволяє суттєво знизити обчислювальну складність операцій з великими числами. Реалізація пакетної обробки підписів та розпаралелювання обчислень на графічних процесорах демонструє значний приріст продуктивності для високонавантажених систем електронного документообігу [29].

Сучасні методи криптоаналізу цифрового підпису включають застосування машинного навчання для пошуку статистичних залежностей в процесі формування підпису. Розробка контрзаходів проти таких атак вимагає впровадження додаткових механізмів рандомізації та маскування операцій з закритим ключем. Дослідження показують, що використання адаптивних алгоритмів захисту дозволяє ефективно протидіяти спробам компрометації системи цифрового підпису через аналіз побічних каналів витоку інформації [30].

Реалізація цифрового підпису в розподілених системах вимагає вирішення додаткових задач синхронізації та узгодження даних між вузлами мережі. Впровадження механізмів консенсусу та розподіленого зберігання ключової інформації підвищує надійність та відмовостійкість системи. Використання блокчейн-технологій для зберігання підписаних документів забезпечує незмінність та простежуваність історії підписів, що особливо важливо для довгострокового архівного зберігання електронних документів [31].

Інтеграція біометричних технологій з системами цифрового підпису відкриває нові можливості для підвищення рівня захисту закритих ключів. Механізми біометричної автентифікації користувача перед виконанням операції підпису знижують ризики несанкціонованого використання криптографічних ключів. Розробка стандартів та протоколів взаємодії біометричних систем з засобами цифрового підпису створює передумови для широкого впровадження цих технологій в практику електронного документообігу [32].

Методи автоматизації процесів управління цифровими підписами в корпоративних системах включають розробку політик безпеки, налаштування workflow та інтеграцію з системами управління ідентифікацією користувачів. Впровадження централізованого моніторингу та аудиту операцій підпису дозволяє своєчасно виявляти та реагувати на порушення встановлених регламентів використання цифрових підписів. Автоматизація процесів оновлення сертифікатів та перевипуску ключів знижує операційні витрати на адміністрування системи [33].

1.3 Вимоги до цифрового підпису

Технічні характеристики сучасних систем цифрового підпису визначаються комплексом базових та додаткових вимог, що забезпечують надійне функціонування механізмів електронного підписання документів. Математичний апарат криптографічних перетворень повинен гарантувати унікальність підпису для кожного документа та неможливість його підробки без знання закритого ключа. Довжина ключів повинна відповідати актуальним рекомендаціям криптографічного співтовариства та забезпечувати захист від можливих атак з використанням найсучасніших обчислювальних ресурсів [35].

Програмно-апаратні засоби генерації цифрового підпису повинні забезпечувати високий рівень ентропії при формуванні випадкових значень, що використовуються в криптографічних протоколах. Як показано в таблиці 1.12, генератори випадкових чисел мають відповідати суворим вимогам щодо якості генерованих послідовностей та швидкості роботи. Механізми захисту від витоку інформації через побічні канали повинні включати часову та енергетичну рандомізацію операцій з криптографічними ключами [36].

Таблиця 1.12 – Основні вимоги до генераторів випадкових чисел

Параметр	Мінімальне значення	Рекомендоване значення
Ентропія (біт/байт)	7.5	7.9
Період послідовності	2^128	2^256
Швидкість генерації (Мбіт/с)	100	1000

Кількість джерел ентропі	2	4

Продуктивність системи цифрового підпису має забезпечувати обробку документів в режимі реального часу з урахуванням пікових навантажень та планового масштабування. Згідно з даними таблиці 1.13, час формування та перевірки підпису повинен відповідати встановленим нормативам залежно від рівня системи. Механізми балансування навантаження та кешування повинні оптимізувати використання обчислювальних ресурсів при груповій обробці документів [37].

Таблиця 1.13 – Вимоги до продуктивності системи цифрового підпису

Характеристика	Доступність системи (%)	Розширений рівень
Час формування підпису (мс)	< 500	< 100
Час перевірки підпису (мс)	< 200	< 50
Кількість підписів/с	> 100	> 1000
Доступність системи (%)	99.9	99.999

Механізми зберігання закритих ключів повинні забезпечувати їх надійний захист від несанкціонованого доступу та копіювання. Використання апаратних модулів безпеки (HSM) та захищених носіїв інформації є обов'язковим для систем підвищеної надійності. Процедури резервного копіювання та відновлення ключової інформації повинні виключати можливість компрометації криптографічних матеріалів [39].

Протоколи взаємодії компонентів системи цифрового підпису повинні забезпечувати конфіденційність та цілісність переданих даних. Використання захищених каналів зв'язку та механізмів взаємної автентифікації запобігає можливості перехоплення та модифікації інформації. Журналювання всіх операцій з цифровими підписами створює повну картину використання системи для подальшого аудиту [40].

Інтерфейси програмування додатків (API) системи цифрового підпису повинні надавати повний набір функцій для інтеграції з зовнішніми системами. Підтримка стандартних протоколів та форматів даних забезпечує сумісність з різними платформами та середовищами розробки. Документація API повинна

містити детальний опис всіх методів та параметрів, а також приклади їх використання [41].

Механізми моніторингу та діагностики повинні забезпечувати оперативне виявлення та локалізацію проблем в роботі системи цифрового підпису. Збір метрик продуктивності та аналіз журналів подій дозволяє прогнозувати потенційні проблеми та планувати профілактичні роботи. Автоматизовані системи сповіщення повинні інформувати адміністраторів про критичні події та відхилення від нормальної роботи [42].

Процедури оновлення програмного забезпечення та зміни конфігурації системи цифрового підпису повинні виконуватися без переривання роботи користувачів. Механізми поетапного розгортання оновлень та можливість швидкого відкату змін мінімізують ризики втрати працездатності системи. Тестування оновлень в ізольованому середовищі дозволяє виявити потенційні проблеми до їх появи в продуктивній системі [43].

Засоби адміністрування системи цифрового підпису повинні надавати гнучкі можливості управління користувачами та їх правами доступу. Підтримка рольової моделі безпеки та механізмів делегування повноважень забезпечує ефективне розмежування доступу до функцій системи. Інтеграція з корпоративними каталогами користувачів спрощує процеси управління обліковими записами [44].

Система повинна забезпечувати можливість довгострокового зберігання підписаних документів з підтримкою механізмів перевірки дійсності підписів протягом всього терміну зберігання. Використання часових міток та додаткових підтверджень забезпечує можливість верифікації цифрових підписів навіть після закінчення терміну дії сертифікатів. Архівні формати зберігання повинні включати всю необхідну інформацію для автономної перевірки підписів [45].

Механізми відновлення після збоїв повинні гарантувати збереження всіх підписаних документів та можливість їх коректної перевірки. Резервне копіювання баз даних та файлових сховищ має виконуватися без переривання роботи користувачів. Процедури відновлення даних повинні бути документовані

та регулярно тестуватися для забезпечення їх ефективності в критичних ситуаціях [46].

Засоби розробки та тестування цифрового підпису повинні включати інструменти для автоматизації процесів перевірки якості програмного коду та виявлення потенційних вразливостей. Використання статичного та динамічного аналізу коду, а також проведення регулярних тестів на проникнення дозволяє підтримувати високий рівень безпеки системи. Документація з розробки та тестування повинна постійно оновлюватися відповідно до змін в системі [47].

1.4 Основні компоненти системи цифрового підпису

Архітектура сучасної системи цифрового підпису складається з множини взаємопов'язаних компонентів, що забезпечують повний цикл операцій з електронними документами. Модульна структура системи, представлена в таблиці 1.14, визначає основні функціональні блоки та їх взаємодію в процесі формування та перевірки цифрових підписів. Кожен компонент реалізує специфічний набір функцій та має стандартизовані інтерфейси для інтеграції з іншими елементами системи [48].

Таблиця 1.14 – Функціональні компоненти системи цифрового підпису

Компонент	Основні функції	Інтерфейси взаємодії
Криптографічне ядро	Генерація ключів, формування та перевірка підпису	PKCS#11, CSP API
Сховище ключів	Захищене зберігання криптографічних матеріалів	KMIP, HSM API
Сервіс підпису	Обробка запитів на підписання	REST API, SOAP
Валідаційний сервіс	Перевірка статусу сертифікатів	OCSP, CRL
Модуль аудиту	Журналювання та моніторинг	Syslog, ELK

Криптографічне ядро системи реалізує базові математичні операції та алгоритми цифрового підпису. Внутрішня структура цього компонента, детально описана в таблиці 1.15, включає спеціалізовані модулі для роботи з різними криптографічними примітивами. Оптимізація швидкодії досягається за рахунок використання апаратного прискорення та ефективних програмних реалізацій криптографічних алгоритмів [49].

Таблиця 1.15 – Структура криптографічного ядра

Модуль	Генератор випадкових чисел	Швидкодія (оп/с)
Генератор ключів	RSA, ECDSA	100-1000
Модуль підпису	RSA-PSS, ECDSA	5000-10000
Геш-функції	SHA-2, SHA-3	>100000
Генератор випадкових чисел	HRNG, PRNG	>1000000

Модуль управління ключами відповідає за генерацію, зберігання та розподіл криптографічних ключів між компонентами системи. Інтеграція з апаратними модулями безпеки (HSM) забезпечує надійний захист закритих ключів та виконання криптографічних операцій в захищеному середовищі. Механізми резервного копіювання та відновлення ключової інформації реалізуються з використанням схем розділення секрету та багатофакторної автентифікації. Підсистема сертифікації забезпечує створення та управління сертифікатами відкритих ключів. Механізми валідації сертифікатів включають перевірку цифрових підписів центрів сертифікації, контроль терміну дії та перевірку статусу відкликання. Протоколи розповсюдження інформації про відкликані сертифікати оптимізовані для мінімізації мережевого трафіку та забезпечення актуальності даних [50, 51].

Сервіс часових міток забезпечує надійну фіксацію часу створення цифрових підписів. Синхронізація з надійними джерелами точного часу та використання криптографічних протоколів підтвердження часу гарантує неможливість маніпуляцій з часовими мітками. Архівне зберігання підписаних документів включає механізми періодичного оновлення часових міток для забезпечення довгострокової дійсності підписів. Компонент форматування та

перетворення даних відповідає за підготовку документів до підписання та формування структурованих контейнерів цифрових підписів. Підтримка різних форматів електронних документів та стандартів цифрового підпису забезпечує сумісність з широким спектром прикладних систем. Механізми валідації вхідних даних та нормалізації формату документів підвищують надійність процесу підписання [52, 53].

Підсистема аудиту та моніторингу забезпечує збір та аналіз інформації про всі операції з цифровими підписами. Механізми виявлення аномалій та підозрілої активності дозволяють оперативно реагувати на потенційні порушення безпеки. Інтеграція з системами управління подіями безпеки (SIEM) забезпечує комплексний аналіз інцидентів та формування звітності. Компонент управління політиками визначає правила використання цифрових підписів та параметри криптографічних алгоритмів. Гнучкі механізми конфігурації дозволяють адаптувати систему до різних сценаріїв використання та вимог безпеки. Автоматизація процесів застосування політик знижує ризики помилок конфігурації та спрощує адміністрування системи [54, 55].

Модуль взаємодії з користувачами надає інтерфейси для виконання операцій підписання та перевірки документів. Підтримка різних механізмів автентифікації та авторизації забезпечує надійний контроль доступу до функцій системи. Інтеграція з системами управління ідентифікацією спрощує процеси управління користувачами та їх повноваженнями. Підсистема зберігання документів забезпечує надійне збереження підписаних файлів та пов'язаних метаданих. Механізми індексації та пошуку дозволяють швидко знаходити необхідні документи за різними критеріями. Контроль цілісності та шифрування даних при зберіганні захищає документи від несанкціонованого доступу та модифікації [56, 57].

Компонент маршрутизації та балансування навантаження оптимізує розподіл запитів між серверами обробки підписів. Механізми масштабування та відмовостійкості забезпечують стабільну роботу системи при зростанні навантаження. Моніторинг продуктивності та автоматичне керування ресурсами підтримують оптимальний режим роботи системи. Модуль інтеграції забезпечує

взаємодію з зовнішніми системами та сервісами. Стандартизовані протоколи та формати обміну даними спрощують процеси інтеграції та підтримки міжсистемної взаємодії. Механізми трансформації форматів та протоколів забезпечують сумісність з різними технологічними платформами [58, 59].

Компонент відновлення та резервного копіювання реалізує механізми захисту від втрати даних та забезпечення безперервності роботи системи. Процедури створення та зберігання резервних копій оптимізовані для мінімізації впливу на продуктивність системи. Регулярне тестування процедур відновлення гарантує можливість швидкого відновлення працездатності в разі збоїв. Підсистема оновлення та управління версіями забезпечує контрольоване впровадження змін в компоненти системи. Механізми автоматичного розгортання оновлень та відкату змін мінімізують ризики порушення роботи системи. Контроль версій програмного забезпечення та конфігурацій підтримує узгодженість компонентів системи [60, 61].

Математична модель криптографічного ядра системи базується на складних алгебраїчних структурах та теорії чисел. Основою більшості сучасних алгоритмів цифрового підпису є властивості еліптичних кривих над скінченними полями. Нехай E - еліптична крива над полем F_q, де q - степінь простого числа. Точки еліптичної кривої утворюють адитивну абелеву групу, в якій складання точок визначається геометрично. Для точки P ∈ E(F_q) порядку n, підгрупа ⟨P⟩ використовується як основа для криптографічних перетворень. Закритий ключ представляється як випадкове число d ∈ [1, n-1], а відповідний відкритий ключ обчислюється як $Q = dP$. Процес формування цифрового підпису для повідомлення m включає наступні етапи: спочатку обчислюється геш-значення h = $H(m)$, де H - криптографічна геш-функція. Далі генерується випадкове число k $\in [1, n-1]$ і обчислюється точка $R = kP = (x\_r, y\_r)$. Значення підпису (r, s) визначається як $r = x\_r \bmod n$ та $s = k^{(-1)}(h + dr) \bmod n$. Верифікація підпису виконується шляхом обчислення точки $R' = (h/s)P + (r/s)Q$ та перевірки умови $x\_R' \bmod n = r$. Реалізація криптографічного ядра вимагає особливої уваги до оптимізації базових операцій в групі точок еліптичної кривої. Операція скалярного множення точки kP є найбільш обчислювально складною і

реалізується з використанням методу подвоєнь та додавань. Для підвищення продуктивності застосовуються різні оптимізації, такі як метод ковзного вікна, NAF-представлення множника та попереднє обчислення часто використовуваних значень [62, 63].

Важливим аспектом реалізації є забезпечення захисту від атак через сторонні канали. Атаки за часом та споживанням енергії можуть дозволити зловмиснику отримати інформацію про секретні ключі шляхом аналізу фізичних характеристик роботи пристрою. Для протидії таким атакам застосовуються методи регуляризації обчислень, випадкове маскування проміжних значень та константний час виконання критичних операцій. Підсистема управління ключами реалізує складний життєвий цикл криптографічних матеріалів. Генерація ключових пар виконується з використанням апаратних генераторів випадкових чисел та включає процедури перевірки якості випадкових послідовностей. Статистичні тести, такі як набір NIST SP 800-22, застосовуються для валідації властивостей випадковості. Згенеровані ключі захищаються за допомогою схем шифрування та розділення секрету. Протокол розділення секрету Шаміра (t, n) дозволяє розділити секретний ключ на n частин таким чином, що для його відновлення потрібно щонайменше t частин. Математично це реалізується шляхом побудови полінома степеня t-1 над скінченним полем, де вільний член є секретним значенням. Кожна частка секрету представляє собою пару (x_i, y_i), де $y\_i = f(x\_i)$ для деякого x_i. Відновлення секрету виконується за допомогою інтерполяції Лагранжа [64, 65, 66].

Механізми оновлення ключів базуються на протоколах безпечного обміну ключами. Протокол Діффі-Геллмана на еліптичних кривих (ECDH) дозволяє двом сторонам узгодити спільний секретний ключ через незахищений канал. Нехай сторони A і B мають пари ключів (d_A, Q_A) та (d_B, Q_B) відповідно. Спільний секрет обчислюється як $K = d\_A * Q\_B = d\_B * Q\_A = d\_A * d\_B * P$. Архітектура сховища ключів забезпечує ієрархічну структуру зберігання криптографічних матеріалів. Майстер-ключі захищаються за допомогою апаратних модулів безпеки (HSM) та використовуються для шифрування робочих ключів. Механізм обгортання ключів (key wrapping) базується на симетричних

алгоритмах шифрування в режимі роботи з автентифікацією. Кожен захищений ключ супроводжується метаданими, що визначають його призначення, термін дії та обмеження використання. Підсистема сертифікації реалізує інфраструктуру відкритих ключів (PKI) з ієрархічною структурою центрів сертифікації. Кореневий центр сертифікації генерує самопідписаний сертифікат та використовується для підписання сертифікатів проміжних центрів [67, 68].

Формат сертифікатів відповідає стандарту X.509 та включає розширення для визначення політик використання та обмежень. Процес верифікації сертифіката включає побудову та перевірку ланцюжка сертифікації до довіреного кореневого сертифіката. Механізми відкликання сертифікатів реалізуються через списки відкликаних сертифікатів (CRL) та онлайн-протокол перевірки статусу сертифіката (OCSP). CRL представляє собою підписаний список серійних номерів відкликаних сертифікатів з причинами відкликання. OCSP-сервер надає актуальну інформацію про статус конкретного сертифіката у реальному часі. Для оптимізації розміру CRL використовуються механізми дельта-CRL та розподілу точок розповсюдження. Сервіс часових міток забезпечує надійну прив'язку цифрових підписів до моменту часу. Математична модель часової мітки включає геш-значення підписаного документа $h = H(m||\sigma)$, де σ - значення цифрового підпису, та точний час t. Часова мітка T формується як цифровий підпис сервера міток над конкатенацією $h||t$. Верифікація часової мітки включає перевірку підпису сервера та валідацію точності зазначеного часу відносно довірених джерел [69, 70, 71].

Механізми довгострокового зберігання підписаних документів враховують можливість компрометації криптографічних алгоритмів з часом. Для забезпечення дійсності підписів у довгостроковій перспективі використовується техніка перепідписання з використанням більш стійких алгоритмів. Процес оновлення включає формування нового підпису над оригінальним документом та всіма попередніми підписами з відповідними часовими мітками. Архітектура системи аудиту базується на розподіленому журналюванні подій з використанням криптографічних механізмів захисту цілісності. Кожен запис журналу включає часову мітку, ідентифікатор джерела, тип події та детальний

опис операції. Механізм зчеплення записів реалізується через включення геш-значення попереднього запису у поточний запис та його підписання. Така структура утворює криптографічно захищений ланцюжок, що запобігає модифікації історичних даних. Аналіз журналів аудиту виконується з використанням методів машинного навчання для виявлення аномальної поведінки. Моделі класифікації навчаються на наборах нормальних та аномальних патернів використання системи. Векторне представлення подій враховує часові характеристики, послідовності операцій та взаємозв'язки між подіями. Виявлені аномалії класифікуються за рівнем критичності та генерують відповідні сповіщення [72, 73].

Підсистема управління політиками реалізує гнучку модель визначення правил використання цифрових підписів. Політики описуються за допомогою формальної мови специфікації з підтримкою логічних операторів та темпоральних умов. Кожне правило включає предикати, що визначають умови застосування, та дії, що мають бути виконані. Механізм розв'язання конфліктів базується на пріоритетах правил та стратегіях комбінування результатів.

Процес застосування політик оптимізований з використанням структур даних для ефективного пошуку відповідних правил. Дерево рішень будується на основі предикатів правил та дозволяє швидко визначати набір застосовних політик для конкретної операції. Кешування результатів обчислення предикатів та проміжних рішень знижує обчислювальні витрати при обробці послідовності схожих запитів. Модуль взаємодії з користувачами реалізує багаторівневу модель автентифікації та авторизації. Механізми автентифікації включають підтримку різних факторів: паролі, сертифікати, токени та біометричні дані. Процес автентифікації використовує протоколи з нульовим розголошенням для доведення володіння секретними даними без їх передачі. Авторизація базується на рольовій моделі доступу з підтримкою ієрархії ролей та делегування повноважень.

Архітектура підсистеми зберігання документів забезпечує високу доступність та масштабованість. Документи зберігаються у розподіленому сховищі з реплікацією даних між вузлами. Механізми узгодження реплік

базуються на алгоритмах консенсусу, що гарантують узгодженість даних при одночасних змінах. Індексація документів виконується з використанням інвертованих індексів та B-дерев для ефективного пошуку за різними атрибутами. Захист документів при зберіганні реалізується через багаторівневу систему шифрування. Симетричне шифрування використовується для захисту вмісту документів, а ключі шифрування захищаються за допомогою асиметричної криптографії. Механізми контролю цілісності базуються на деревах Меркла, що дозволяють ефективно виявляти модифікації даних у великих наборах документів. Компонент маршрутизації реалізує динамічне балансування навантаження з урахуванням стану серверів та характеристик запитів. Алгоритми маршрутизації враховують латентність мережі, завантаженість процесора та пам'яті, а також специфічні метрики продуктивності криптографічних операцій. Механізми виявлення та обходу відмов забезпечують безперервність обслуговування при збоях окремих компонентів.

Масштабування системи виконується через механізми горизонтального розширення з додаванням нових вузлів обробки. Процес масштабування включає автоматичне налаштування розподілу навантаження та реплікації даних. Моніторинг продуктивності базується на зборі та аналізі метрик у реальному часі з використанням часових рядів та прогнозного моделювання. Інтеграційний модуль реалізує підтримку різних протоколів та форматів обміну даними. Трансформація форматів виконується з використанням проміжного канонічного представлення, що спрощує додавання підтримки нових форматів. Механізми валідації вхідних даних включають перевірку схем та семантичну валідацію змісту документів.

Підсистема резервного копіювання реалізує інкрементальне резервне копіювання з дедуплікацією даних. Процес резервного копіювання оптимізований для мінімізації впливу на продуктивність системи через використання механізмів знімків та паралельної обробки. Процедури відновлення включають автоматичну верифікацію цілісності відновлених даних та їх узгодженості. Механізми оновлення системи забезпечують можливість оновлення компонентів без переривання роботи. Процес оновлення включає

автоматичну міграцію даних та конфігурацій до нових версій. Відкат змін у разі виявлення проблем виконується автоматично на основі заздалегідь визначених критеріїв успішності оновлення.

2. АЛГОРИТМИ ЦИФРОВОГО ПІДПИСУ НА ОСНОВІ ГЕШ-ФУНКЦІЙ

2.1 Загальні принципи використання геш-функцій у цифрових підписах

Геш-функції являють собою математичні алгоритми, які перетворюють вхідні дані довільного розміру у вихідний бітовий рядок фіксованої довжини. При використанні в системах цифрового підпису геш-функції вирішують проблему обробки великих обсягів даних, оскільки замість підписування всього повідомлення виконується підпис його геш-значення. Процес формування геш-значення передбачає розбиття вхідного повідомлення на блоки однакового розміру та послідовне застосування до них раундових перетворень з використанням ключової інформації.

Процес формування геш-значення зображений на рисунку 2.1 який демонструє послідовність перетворень вхідних даних. Односторонність геш-функції забезпечується складністю математичних операцій та використанням незворотних перетворень в раундових функціях.

Рисунок 2.1. – Алгоритм формування геш-значення повідомлення

Для криптографічно стійких геш-функцій обчислювально складно знайти прообраз - вхідне повідомлення, що дає задане геш-значення. Ця властивість гарантує, що зловмисник не зможе підібрати повідомлення з таким самим геш-значенням для підробки цифрового підпису.

Стійкість до колізій першого роду означає складність знаходження двох різних повідомлень з однаковим геш-значенням. Стійкість до колізій другого роду передбачає складність для заданого повідомлення знайти інше повідомлення з тим самим значенням геш-функції. Обидві властивості критично важливі для систем цифрового підпису, оскільки наявність колізій дозволила б зловмиснику створювати підроблені підписи шляхом заміни оригінального повідомлення на колізійне.

Сучасні криптографічні геш-функції будуються на основі ітеративної конструкції Меркла-Дамгарда або губки. Конструкція Меркла-Дамгарда передбачає розбиття вхідного повідомлення на блоки фіксованого розміру та послідовне застосування до них стискаючої функції (Див. Рис. 2.3). Кожен раунд стискання приймає на вхід поточний блок повідомлення та результат обробки попереднього блоку. Остання ітерація дає фінальне геш-значення.

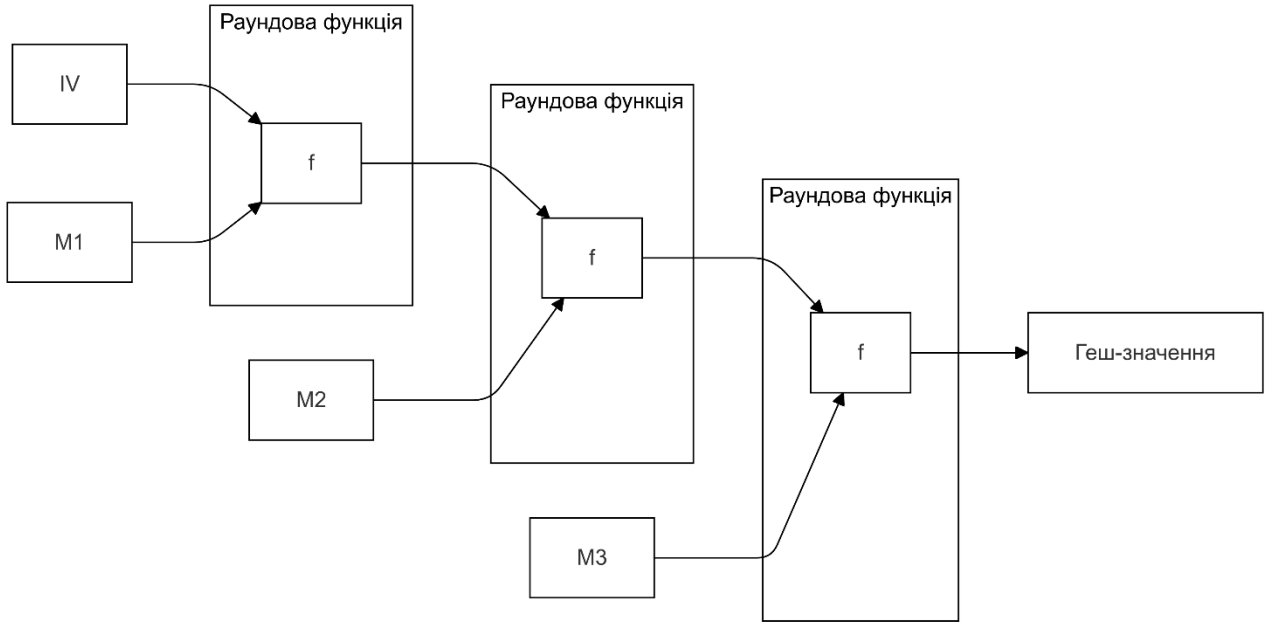

Рисунок 2.3 – Схема роботи Меркла-Дамгарда

На практиці найбільш поширеними є геш-функції сімейства SHA (Secure Hash Algorithm). SHA-1 формує геш-значення довжиною 160 біт, але через

знайдені вразливості не рекомендується для нових застосувань. SHA-2 включає функції SHA-224, SHA-256, SHA-384 та SHA-512, що відрізняються довжиною геш-значення та розміром оброблюваних блоків. SHA-3 базується на конструкції губки та забезпечує кращу продуктивність на апаратних реалізаціях.

При використанні в схемах цифрового підпису геш-функції дозволяють зменшити обсяг обчислень, оскільки криптографічні операції виконуються не над самим повідомленням, а над його геш-значенням фіксованої довжини. Геш-значення також служить способом однозначної ідентифікації повідомлення - будь-яка модифікація вхідних даних призводить до зміни геш-значення з високою ймовірністю.

Механізм формування цифрового підпису передбачає обчислення геш-значення повідомлення та його подальше шифрування закритим ключем підписувача (Див. Рис. 2.3). Перевірка підпису включає розшифрування підпису відкритим ключем та порівняння отриманого значення з незалежно обчисленим гешем повідомлення. Збіг значень підтверджує справжність підпису та цілісність даних.

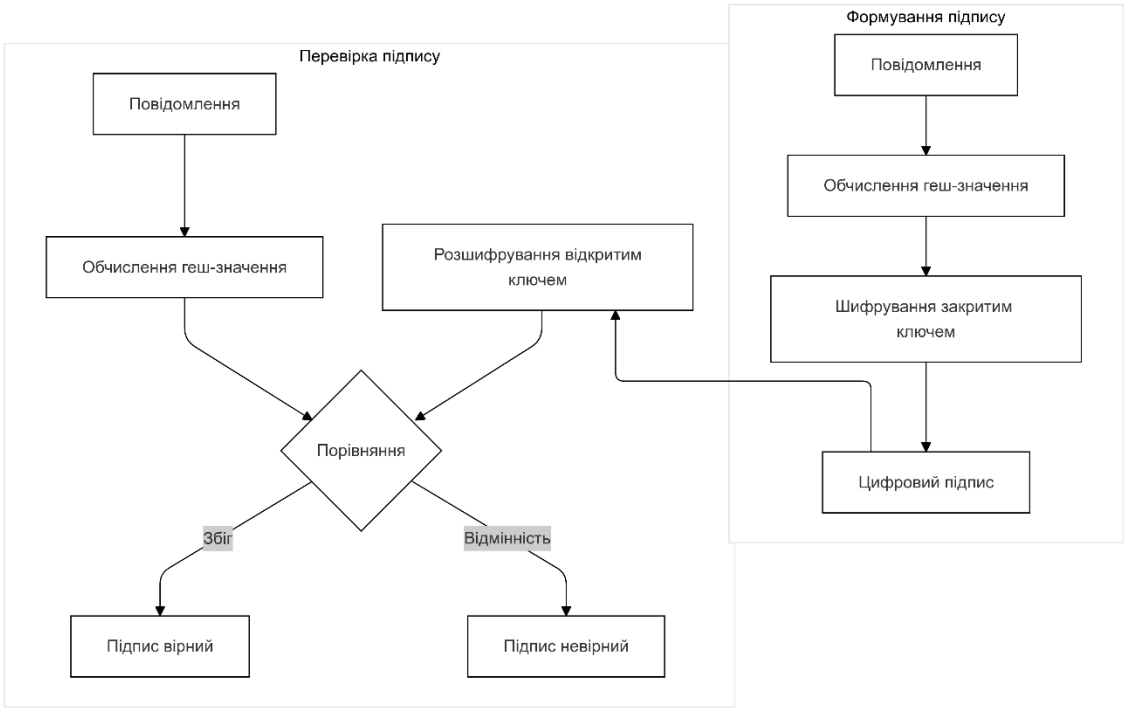

Рисунок 2.4 – Схема формування та перевірки цифрового підпису

Застосування геш-функцій в схемах цифрового підпису забезпечує низку переваг. Зменшення розміру підпису покращує ефективність зберігання та передачі підписаних документів. Фіксована довжина геш-значення спрощує реалізацію криптографічних алгоритмів та зменшує ймовірність помилок. Властивості односторонності та стійкості до колізій гарантують неможливість підробки підпису шляхом модифікації повідомлення або пошуку колізій.

Безпека схем цифрового підпису значною мірою залежить від криптографічної стійкості використовуваної геш-функції. Компрометація геш-функції, наприклад знаходження ефективного способу побудови колізій, може призвести до можливості підробки підписів. Розвиток квантових обчислень створює нові виклики для криптографічних геш-функцій, оскільки квантові алгоритми можуть прискорити пошук колізій.

Розробка постквантових геш-функцій є актуальним напрямком досліджень. Перспективними вважаються конструкції на основі решіток, багатовимірних квадратичних систем та інших математичних проблем, для яких не відомо ефективних квантових алгоритмів. Паралельно ведеться робота над покращенням продуктивності існуючих геш-функцій та зменшенням їх ресурсоємності для використання на мобільних та вбудованих пристроях.

2.2 Алгоритм створення цифрового підпису на основі геш-функцій

Цифровий підпис на основі геш-функцій реалізується через послідовність математичних перетворень, які забезпечують цілісність та автентичність електронних документів. Процес починається з обробки вихідного повідомлення M за допомогою криптографічної геш-функції, яка генерує фіксований за розміром відбиток даних. Геш-функція перетворює вхідні дані довільної довжини у вихідне значення фіксованого розміру, зазвичай 256 або 512 біт.

$$H(M) = Hash(M) \tag{5}$$

При створенні цифрового підпису використовуються властивості геш-функцій, зокрема незворотність та стійкість до колізій. Незворотність означає

неможливість відновлення вихідного повідомлення за його гешем, а стійкість до колізій гарантує, що ймовірність знаходження двох різних повідомлень з однаковим значенням гешу є мізерно малою. Для надійного гешування застосовуються алгоритми SHA-256, SHA-512 або SHA-3, які пройшли серйозне криптографічне тестування.

Наступним етапом створення цифрового підпису є шифрування отриманого геш-значення H(M) за допомогою приватного ключа відправника SK. Процес шифрування виконується з використанням асиметричного криптографічного алгоритму, найчастіше RSA або алгоритмів на еліптичних кривих.

$$S = EncryptSK(H(M)) \quad (6)$$

Приватний ключ SK зберігається в секреті і використовується виключно власником для створення підписів. Процес шифрування гешу приватним ключем створює цифровий підпис S, який є унікальним для кожного повідомлення та власника ключа. Розмір підпису залежить від використовуваного алгоритму шифрування та довжини ключа.

Механізм формування цифрового підпису включає генерацію випадкових чисел для забезпечення унікальності кожного підпису. Навіть при повторному підписанні того самого повідомлення тим самим ключем, значення підпису буде відрізнятися через використання випадкових параметрів. Це забезпечує додатковий рівень захисту від атак повторного відтворення.

При використанні алгоритму RSA для створення цифрового підпису застосовується модульне експоненціювання, де геш-значення повідомлення піднімається до степеня, що відповідає приватному ключу, за модулем великого складеного числа. Параметри RSA вибираються таким чином, щоб забезпечити достатній рівень криптографічної стійкості, зазвичай використовуються ключі довжиною 2048 або 4096 біт.

Схема цифрового підпису на основі еліптичних кривих (ECDSA) використовує операції на еліптичній кривій для створення підпису. ECDSA забезпечує той самий рівень безпеки, що й RSA, але з меншою довжиною ключа.

Для досягнення рівня безпеки, еквівалентного RSA-2048, достатньо використовувати ключі ECDSA довжиною 256 біт.

Математична модель створення цифрового підпису включає операції в кінцевих полях та групах точок еліптичної кривої. Для ECDSA підпис складається з двох компонентів (r, s), які обчислюються з використанням випадкового числа k, приватного ключа d та геш-значення повідомлення H(M). Компоненти підпису обчислюються за формулами, що включають операції множення точки еліптичної кривої на скаляр та модульні арифметичні операції. Еліптична крива в ECDSA — це лінія на площині, що задається рівнянням $y^2=x^3+a\cdot x+b$, де a та b — такі числа, що $4\cdot a^3+27\cdot b^2\neq 0$. Наприклад, Bitcoin та Ethereum використовують криву $y^2=x^3+7$ (Рис. 2.4).

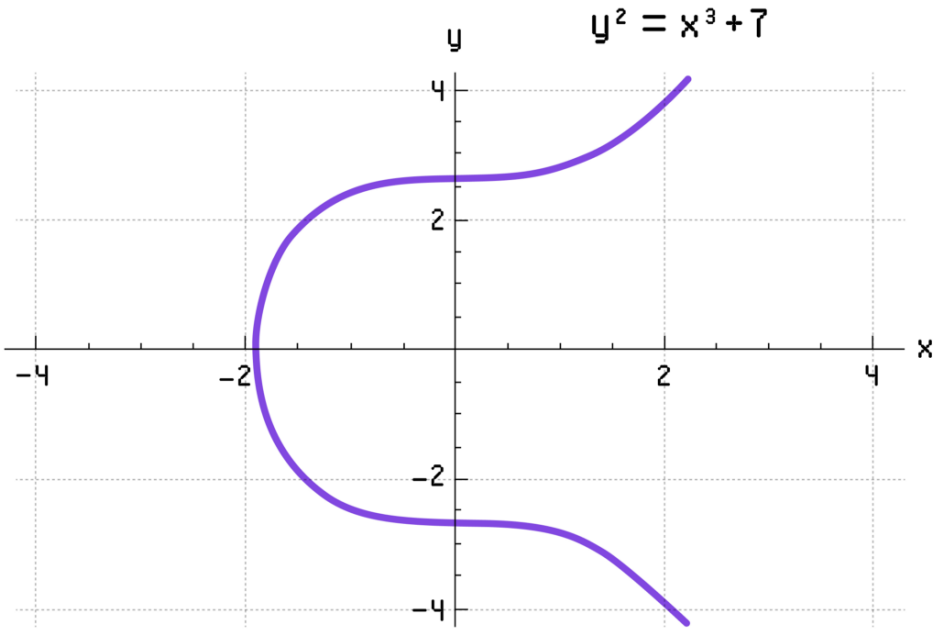

Рисунок 2.4 – Еліптична крива ECDSA

При реалізації алгоритму цифрового підпису особлива роль відводиться генератору випадкових чисел. Якість випадкових чисел безпосередньо впливає на безпеку схеми підпису. Для генерації випадкових чисел використовуються апаратні генератори випадкових чисел або криптографічно стійкі псевдовипадкові генератори, які забезпечують непередбачуваність та рівномірний розподіл генерованих значень.

Процес верифікації цифрового підпису виконується з використанням відкритого ключа PK, який математично пов'язаний з приватним ключем SK. Верифікація включає розшифрування підпису відкритим ключем та порівняння отриманого значення з гешем повідомлення. Коректність верифікації гарантується математичними властивостями використовуваних криптографічних алгоритмів.

Реалізація алгоритму цифрового підпису вимагає ретельного вибору криптографічних примітивів та їх параметрів. Довжина ключів, параметри еліптичної кривої, характеристики геш-функції повинні забезпечувати необхідний рівень криптографічної стійкості з урахуванням сучасних обчислювальних можливостей та відомих методів криптоаналізу.

Для забезпечення максимальної безпеки при створенні цифрового підпису використовуються стандартизовані криптографічні примітиви та протоколи. Стандарти цифрового підпису, такі як DSS (Digital Signature Standard) та ГОСТ Р 34.10-2012, визначають вимоги до реалізації алгоритмів підпису та форматів представлення даних.

Обчислювальна складність алгоритму створення цифрового підпису залежить від розміру повідомлення та використовуваних криптографічних параметрів. Основні обчислювальні витрати припадають на операції модульного експоненціювання в RSA або операції на еліптичній кривій в ECDSA. Оптимізація цих операцій є важливим аспектом практичної реалізації схеми цифрового підпису.

При програмній реалізації алгоритму цифрового підпису необхідно враховувати особливості роботи з великими числами та забезпечувати коректне виконання модульних арифметичних операцій. Бібліотеки криптографічних функцій надають оптимізовані реалізації необхідних алгоритмів та операцій з урахуванням особливостей цільової платформи.

Надійність схеми цифрового підпису базується на складності розв'язання математичних задач факторизації великих чисел та дискретного логарифмування. Безпека RSA ґрунтується на складності факторизації великих

складених чисел, а безпека ECDSA - на складності обчислення дискретного логарифма в групі точок еліптичної кривої.

2.3 Алгоритм перевірки цифрового підпису

Алгоритм перевірки цифрового підпису починається з обробки отриманого повідомлення за допомогою криптографічного алгоритму гешування. Система використовує ту саму геш-функцію, що застосовувалась при створенні підпису, для генерації геш-значення отриманого повідомлення. Процес гешування забезпечує створення унікального цифрового відбитка фіксованої довжини для будь-якого вхідного повідомлення.

Таблиця 2.1 – Основні параметри верифікації цифрового підпису

Параметр	Опис	Формат
Повідомлення (M)	Вхідні дані для перевірки	Довільний формат
Цифровий підпис (S)	Підпис для верифікації	Бінарні дані
Публічний ключ (PK)	Ключ для розшифрування	ASN.1 DER
Геш оригіналу H(M)	Геш отриманого повідомлення	256/512 біт
Геш з підпису H'(M)	Розшифрований геш	256/512 біт

Протокол перевірки цифрового підпису передбачає порівняння двох геш-значень: одного, отриманого безпосередньо з повідомлення, та іншого, відновленого з цифрового підпису. Рівність цих значень служить математичним доказом автентичності підпису та цілісності повідомлення. Розбіжність значень вказує на порушення цілісності даних або підробку підпису.

Таблиця 2.2 – Можливі результати верифікації

Результат	Причина	Дії
Підпис валідний	H(M) = H'(M)	Прийняти повідомлення
Підпис невалідний	H(M) ≠ H'(M)	Відхилити повідомлення
Помилка формату	Некоректні вхідні дані	Запит повторної передачі
Помилка ключа	Невідповідний ключ	Перевірка сертифіката
Збій алгоритму	Внутрішня помилка	Діагностика системи

Процес верифікації при використанні алгоритму RSA базується на властивостях модульної арифметики та теоремі Ейлера. Розшифрування підпису включає піднесення значення підпису до степеня відкритого ключа за модулем того ж складеного числа, що використовувалось при створенні підпису. Математична коректність цього процесу гарантується властивостями функції Ейлера.

Безпека процесу верифікації підпису залежить від стійкості використовуваних криптографічних примітивів. Геш-функції повинні забезпечувати стійкість до колізій та прообразів, а асиметричні алгоритми - складність розв'язання відповідних математичних задач факторизації або дискретного логарифмування.

Реалізація перевірки підпису повинна включати механізми захисту від різних типів атак. Система повинна перевіряти коректність форматів даних, валідність параметрів криптографічних алгоритмів, відповідність часових міток та версій протоколів. Додаткові перевірки підвищують загальну надійність системи.

Продуктивність процесу верифікації залежить від ефективності реалізації криптографічних операцій. Оптимізація включає використання швидких алгоритмів модульної арифметики, попередні обчислення для операцій на еліптичних кривих, ефективну реалізацію геш-функцій. Баланс між швидкодією та безпекою досягається вибором оптимальних параметрів.

Інтеграція механізму перевірки підпису в прикладні системи вимагає розробки зручних програмних інтерфейсів. API повинні надавати методи для

виконання верифікації, обробки помилок та отримання результатів у форматі, придатному для подальшої обробки. Документація повинна чітко описувати всі аспекти використання API.

На рисунку 2.1 відображено алгоритм перевірки цифрового підпису.

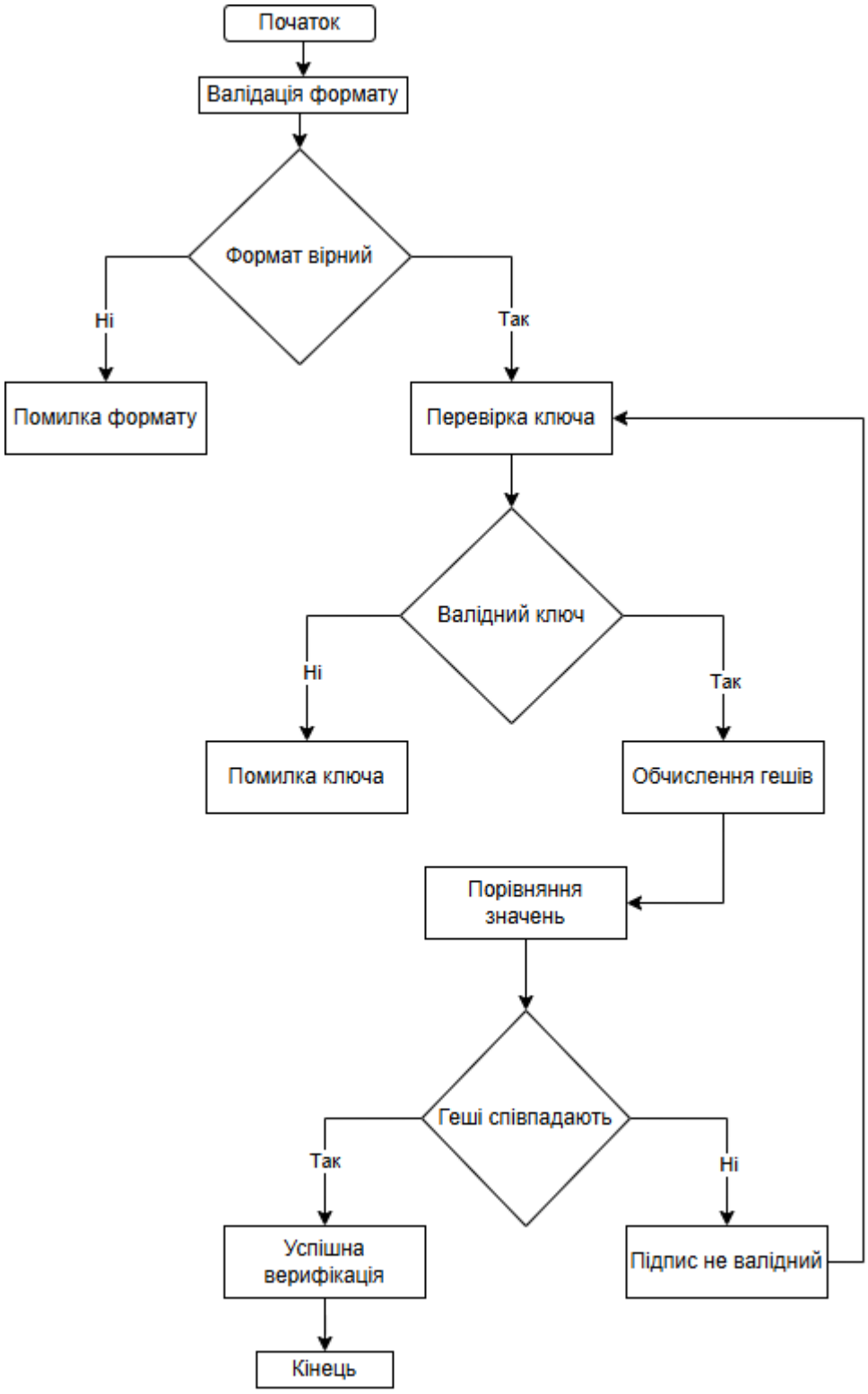

Рисунок 2.5. – Алгоритм перевірки цифрового підпису.

Перевірка підпису на основі алгоритму ECDSA вимагає виконання специфічних операцій на еліптичній кривій. Процес включає обчислення точки кривої з використанням компонентів підпису та порівняння її координати з відповідним значенням у підписі. Коректність підпису підтверджується при співпадінні обчислених значень.

Тестування системи верифікації підпису включає перевірку коректності роботи на різних наборах вхідних даних. Тести повинні охоплювати стандартні випадки використання, обробку помилкових даних, спроби атак та граничні випадки параметрів алгоритмів. Результати тестування документуються для подальшого аналізу.

Документування процесу перевірки підпису повинно включати детальний опис алгоритму, форматів даних, послідовності операцій та можливих помилок. Чітка документація спрощує впровадження та підтримку системи, допомагає користувачам правильно інтегрувати функціональність верифікації підпису.

Стандартизація процесу перевірки цифрового підпису забезпечує сумісність різних реалізацій та гарантує необхідний рівень безпеки. Відповідність міжнародним та галузевим стандартам підтверджується через процедури сертифікації криптографічних модулів та програмних рішень.

Практичне застосування механізму верифікації підпису вимагає врахування специфіки конкретного середовища експлуатації. Система повинна коректно працювати в умовах різних операційних систем, апаратних платформ та мережевих конфігурацій. Підтримка різних форматів даних та протоколів розширює можливості застосування.

2.4 Алгоритми цифрового підпису, засновані на геш-функціях

DSA є одним з основних стандартів цифрового підпису, розробленим Національним інститутом стандартів і технологій США. Алгоритм використовує обчислення в кінцевому полі та базується на складності задачі дискретного логарифмування. Процес створення підпису включає генерацію випадкового числа k та обчислення двох компонентів підпису r і s. DSA спочатку

використовував геш-функцію SHA-1, але пізніші версії стандарту перейшли на SHA-256 для підвищення криптографічної стійкості.

Таблиця 2.3. – Порівняння алгоритмів цифрового підпису

Алгоритм	Геш-функція	Розмір ключів (біт)	Сфера застосування
DSA	SHA-1, SHA-256	1024-3072	PKI, сертифікати X.509
ECDSA	SHA-256, SHA-384	256-384	Криптовалюти, TLS
EdDSA	SHA-512	256	Signal, SSH, TOR
RSA-PSS	SHA-256, SHA-512	2048-4096	Корпоративні системи
SPHINCS+	SHA-256, SHAKE256	128-256	Постквантова криптографія

ECDSA представляє модифікацію класичного DSA з використанням арифметики еліптичних кривих. Алгоритм застосовує точки на еліптичній кривій замість елементів кінцевого поля, що дозволяє досягти того ж рівня безпеки при значно менших розмірах ключів. ECDSA широко застосовується в криптовалютах, де Bitcoin використовує криву secp256k1, а багато інших проектів - криву Curve25519.

EdDSA є відносно новим алгоритмом підпису, розробленим Деніелом Бернштейном та його колегами. Алгоритм базується на кривій Edwards25519 та використовує геш-функцію SHA-512 для генерації детермінованих підписів. EdDSA забезпечує високу швидкість операцій, стійкість до атак побічними каналами та детерміновану генерацію підписів без необхідності використання генератора випадкових чисел. На рисунку 2.6 зображена криптографічна еліптична крива яка забезпечує 128-бітне шифрування при розмірі ключа 256-біт, призначенна для викоистання зі схемою узгодження ключів Діффі - Хеллмана (ECDH).

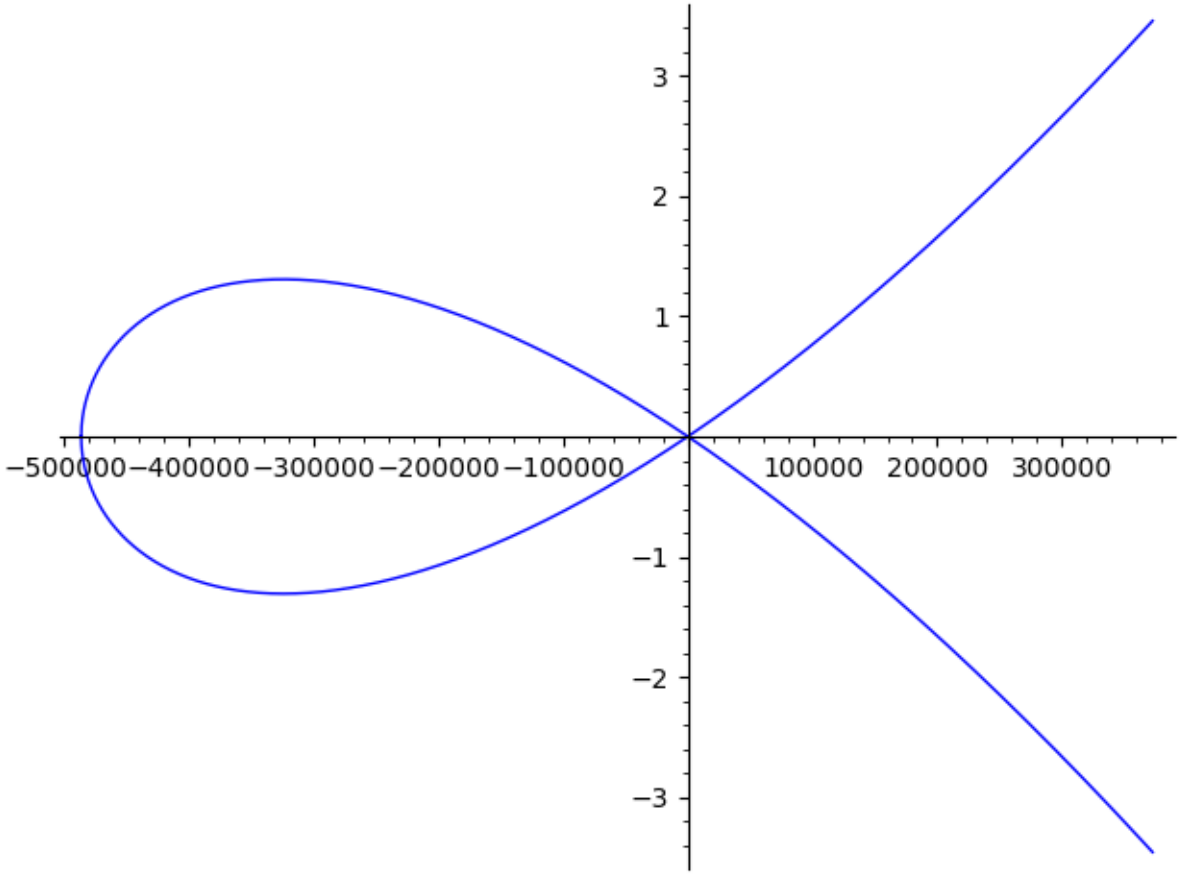

Рисунок 2.6. – Графік криптографічної еліптичної кривої Curve25519

RSA-PSS представляє вдосконалену схему підпису RSA з використанням імовірнісного заповнення. Алгоритм використовує геш-функції SHA-256 або SHA-512 та включає додаткові етапи обробки даних для підвищення безпеки. RSA-PSS забезпечує доказову безпеку та широко застосовується в корпоративних системах та інфраструктурі відкритих ключів.

Методи реалізації алгоритмів цифрового підпису вимагають оптимізації критичних операцій. Для ECDSA основними оптимізаціями є швидкі алгоритми множення точок еліптичної кривої та ефективна реалізація модульної арифметики. EdDSA використовує спеціально підібрані параметри кривої Edwards25519 для максимальної продуктивності операцій.

Ed25519 — схема цифрового підпису EdDSA, яка використовує SHA-512 і еліптичну криву, пов'язану з Curve25519. Основні характеристики схеми:

$q = 2^{255} - 19$

E / F_q — еліптична крива Едвардса:

$$-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2 \quad (7)$$

$\ell = 2^{252} + 27742317777372353535851937790883648493$ — порядок точки B на кривій $E(F_q)$.

B — унікальна точка $E(F_q)$, у якої у-координата дорівнює 4/5, а х-координата — додатна (якщо говорити в термінах бітового кодування).

H – SHA-512, с b = 256.

Крива $E(F_q)$ біраціонально еквівалентна кривій Монтгомері, відомій як edwards25519. Еквівалентність виражається формулами:

$$x = \frac{u}{v}, y = \frac{u-1}{u+1} \quad (8)$$

де −486664 використовується як один із параметрів у рівняннях.

Вибір геш-функції для алгоритму підпису впливає на загальну безпеку схеми. SHA-256 забезпечує 128 біт безпеки та широко використовується в сучасних реалізаціях. SHA-512 надає вищий рівень безпеки та краще підходить для 64-бітових архітектур. Нові алгоритми можуть використовувати функції сімейства SHA-3 або спеціалізовані геш-функції.

Специфіка застосування алгоритмів підпису в різних областях визначає додаткові вимоги до їх реалізації. Криптовалюти потребують швидкої перевірки великої кількості підписів, що робить ECDSA та EdDSA оптимальним вибором. Корпоративні системи часто використовують RSA-PSS через його сумісність з існуючою інфраструктурою.

Розробка нових алгоритмів цифрового підпису враховує загрози з боку квантових комп'ютерів. SPHINCS+ є прикладом постквантового алгоритму підпису, який базується виключно на геш-функціях та залишається безпечним навіть при наявності квантового комп'ютера. Алгоритм використовує деревоподібні структури та множинні геш-обчислення.

Стандартизація алгоритмів цифрового підпису забезпечує їх широке впровадження та сумісність різних реалізацій. NIST публікує стандарти та рекомендації щодо використання алгоритмів підпису, включаючи вимоги до

довжини ключів та вибору параметрів. Міжнародні стандарти визначають формати даних та протоколи взаємодії.

Практична реалізація алгоритмів підпису вимагає використання криптографічних бібліотек. OpenSSL надає реалізації DSA, ECDSA та EdDSA, оптимізовані для різних платформ. Спеціалізовані бібліотеки, такі як libsodium, фокусуються на високорівневих інтерфейсах та безпечних за замовчуванням налаштуваннях.

Тестування реалізацій алгоритмів підпису включає перевірку на відомих тестових векторах. Набори тестових даних дозволяють перевірити коректність реалізації всіх етапів створення та перевірки підпису. Додаткові тести перевіряють обробку помилкових даних та граничних випадків.

Документування особливостей реалізації алгоритмів підпису є критичним для їх правильного використання. Специфікації повинні включати опис форматів даних, обмеження на параметри та рекомендації щодо безпечного застосування. Приклади коду та типові сценарії використання спрощують інтеграцію алгоритмів у прикладні системи.

Майбутній розвиток алгоритмів цифрового підпису спрямований на підвищення ефективності та безпеки. Дослідження включають розробку нових постквантових схем, оптимізацію існуючих алгоритмів та створення спеціалізованих варіантів для конкретних застосувань. Стандартизація нових алгоритмів забезпечує їх широке впровадження.

Еволюція алгоритмів підпису відображає зміни в потребах та загрозах безпеки. Перехід від DSA до ECDSA демонструє важливість оптимізації розміру ключів та швидкодії. Поява EdDSA показує тренд до спрощення реалізації та підвищення захисту від практичних атак.

3. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ ЦИФРОВОГО ПІДПИСУ НА ОСНОВІ ГЕШ-ФУНКЦІЙ

3.1 Архітектурне проєктування системи та вибір технологій

Проектування системи цифрового підпису розпочалось з аналізу доступних технологій та інструментів розробки. Python був обраний як основна мова програмування через його потужні криптографічні бібліотеки та зручність роботи з байтовими даними. Ключовою перевагою Python є наявність бібліотеки cryptography, яка надає низькорівневий доступ до криптографічних примітивів та реалізує сучасні стандарти безпеки. Додатково використовується бібліотека hashlib для обчислення геш-значень, що є невід'ємною частиною процесу цифрового підписування. Архітектура системи базується на об'єктно-орієнтованому підході, що дозволяє створити гнучку та масштабовану структуру коду. На рисунку 3.1 зображено використані бібліотеки та модулі для цієї програми.

```
import hashlib
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.exceptions import InvalidSignature
from functools import lru_cache
import multiprocessing
import time
import os
import base64
import logging
import json
from datetime import datetime
from typing import Optional, Tuple
import platform
import psutil
```

Рисунок 3.1. – Використані бібліотеки для розробки

Розроблена система складається з трьох основних компонентів: модуля управління ключами, модуля підписування та верифікації, а також модуля

тестування продуктивності. Модуль управління ключами реалізований у класі SignatureProcessor, який відповідає за генерацію та зберігання криптографічних ключів. Вибір RSA алгоритму з розміром ключа 2048 біт забезпечує надійний рівень захисту, що відповідає сучасним рекомендаціям з криптографічної стійкості. Реалізація включає використання схеми підпису PSS (Probabilistic Signature Scheme), яка додає елемент випадковості для підвищення безпеки.

```
class SignatureProcessor:
    def __init__(self, key_size: int = 2048):
        self.key_size = key_size
        self._hash_algorithm = hashes.SHA256()
        self._hash_cache = {}
        logger.info(f"Initialized SignatureProcessor with key size: {key_size}")
```

Система логування відіграє важливу роль у відстеженні роботи системи та діагностиці помилок. Реалізація логування включає запис інформації як у файл, так і виведення в консоль, що полегшує розробку та налагодження.

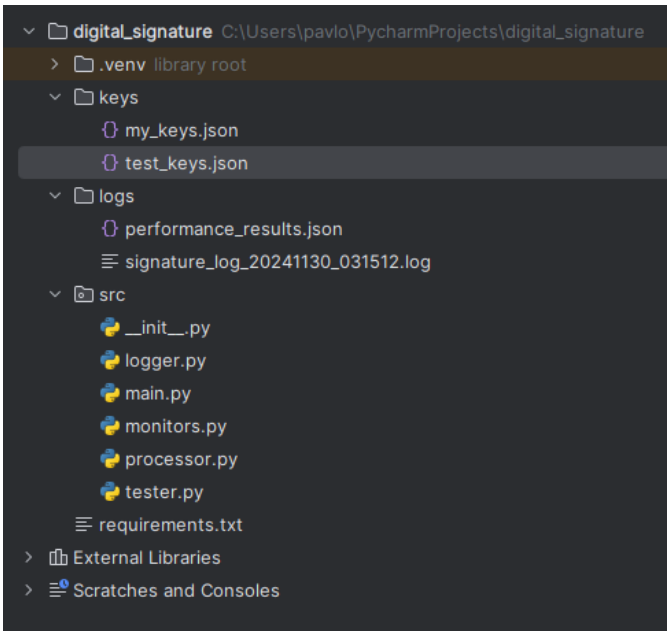

Рисунок 3.2. – Структура проекту

Формат логів містить часову мітку, рівень важливості та детальний опис

події. Логування налаштоване через окрему функцію setup_logging(), яка створює необхідну структуру директорій та встановлює формати виводу.

Моніторинг продуктивності реалізований через окремий клас PerformanceMonitor, який збирає метрики про використання системних ресурсів. Збір даних про використання процесора, пам'яті та кількість потоків дозволяє оптимізувати роботу системи та виявляти потенційні проблеми з продуктивністю. Бібліотека psutil використовується для отримання системної інформації, а multiprocessing забезпечує можливість паралельної обробки даних. На рисунку 3.3 зображено вивід інформації в консоль про генерацію ключа.

```
2024-11-30 02:47:08,199 - INFO - Starting digital signature system testing...
2024-11-30 02:47:08,200 - INFO - Initialized SignatureProcessor with key size: 2048
2024-11-30 02:47:08,200 - INFO - Starting key pair generation...
2024-11-30 02:47:08,275 - INFO - Key pair generated successfully:
                Generation time: 0.07 seconds
                Memory usage: 47.53 MB
                CPU usage: 0.0%
2024-11-30 02:47:08,276 - INFO - Keys successfully saved to keys\my_keys.json
2024-11-30 02:47:08,277 - INFO - Keys successfully loaded from keys\my_keys.json
2024-11-30 02:47:08,277 - INFO - Keys were generated at: 2024-11-30T02:47:08.275775
2024-11-30 02:47:08,313 - INFO - Test message signature verification: True
2024-11-30 02:47:08,314 - INFO - Initialized SignatureProcessor with key size: 2048
2024-11-30 02:47:08,314 - INFO - Starting performance tests...
2024-11-30 02:47:08,487 - INFO -
Testing with message size: 1024 bytes
```

Рисунок 3.3. – Відображення логування в консоль

Обробка великих обсягів даних оптимізована через використання поблокового читання та кешування результатів. Метод _compute_hash реалізує ефективне обчислення геш-значень з використанням декоратора lru_cache для кешування результатів частих операцій. Розмір блоку в 8 КБ вибраний як оптимальний для більшості сценаріїв використання, забезпечуючи баланс між швидкістю роботи та використанням пам'яті.

```
@lru_cache(maxsize=128)
def _compute_hash(self, message: bytes) -> bytes:
    start_time = time.time()
```

hasher = hashlib.sha256()

chunk_size = 8192

Безпека системи забезпечується використанням сучасних криптографічних примітивів та правильним налаштуванням параметрів. Схема підпису PSS з максимальною довжиною солі забезпечує додатковий захист від атак на підпис. Обробка помилок реалізована через систему винятків Python, що дозволяє коректно обробляти нестандартні ситуації та запобігати витоку конфіденційної інформації через повідомлення про помилки.

Серіалізація даних реалізована з використанням форматів PEM для криптографічних ключів та JSON для зберігання додаткової інформації. Формат PEM забезпечує стандартизований спосіб зберігання криптографічних об'єктів, а JSON надає зручний формат для метаданих та результатів тестування. Base64 кодування використовується для безпечного перетворення бінарних даних у текстовий формат при збереженні у JSON.

Паралельна обробка даних реалізована з використанням модуля multiprocessing, що дозволяє ефективно використовувати всі доступні ядра процесора. Клас PerformanceTester включає можливість паралельного виконання операцій підписування та верифікації, що особливо корисно при обробці великої кількості повідомлень. Кількість процесів автоматично адаптується до кількості доступних ядер процесора.

Механізми кешування впроваджені на різних рівнях системи для оптимізації продуктивності. Декоратор lru_cache застосований до методу _compute_hash, що дозволяє уникнути повторних обчислень геш-значень для однакових повідомлень. Розмір кешу встановлений на 128 елементів, що є компромісом між використанням пам'яті та швидкістю роботи. Додатково реалізоване кешування проміжних результатів криптографічних операцій через внутрішній словник _hash_cache.

Обробка вхідних даних включає механізми валідації та нормалізації. Перед підписанням повідомлення конвертуються у байтовий формат з використанням UTF-8 кодування, що забезпечує коректну обробку текстових даних різними

мовами. Розмір вхідних повідомлень перевіряється для запобігання проблем з пам'яттю при обробці надто великих файлів. Великі повідомлення обробляються частинами, що дозволяє працювати з файлами будь-якого розміру.

Механізм роботи з криптографічними ключами передбачає можливість їх оновлення та ротації. Генерація нових ключів може відбуватися за розкладом або при досягненні певних умов, наприклад, після визначеної кількості використань. Старі ключі зберігаються протягом налаштованого періоду для можливості перевірки раніше створених підписів. Процес генерації ключів включає збереження метаданих, таких як дата створення та призначення ключа.

Тестування продуктивності реалізовано з використанням різних розмірів повідомлень для оцінки масштабованості системи. Тестові сценарії включають перевірку швидкодії на повідомленнях розміром від 1 КБ до 100 КБ, що дозволяє оцінити ефективність системи в різних умовах використання. Результати тестів зберігаються у структурованому форматі JSON для подальшого аналізу та візуалізації. На рисунку 3.4 показано збереженя тестової інформації про прудуктивність в формат JSON.


```
{
    "system_info": {
        "platform": "Windows-11-10.0.22631-SP0",
        "processor": "AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD",
        "cpu_count": 12,
        "memory_total_gb": 31.362815856933594,
        "python_version": "3.13.0"
    },
    "tests": [
        {
            "message_size_bytes": 1024,
            "key_generation_time": 0.04149460792541504,
            "signing_time": 1.7326915264129639,
            "verification_time": 0.003271341323852539,
            "memory_usage_mb": 51.7109375,
            "cpu_percent": 0.0,
            "all_signatures_valid": true
        },
        {
```


Рисунок 3.4. – Збереження метриків в формат JSON

Системні метрики збираються та аналізуються для виявлення потенційних проблем з продуктивністю. Моніторинг включає відстеження використання процесора, пам'яті та дискового простору. Результати аналізу використовуються

для оптимізації параметрів системи та планування масштабування. Збір метрик реалізований через клас PerformanceMonitor, який надає статичні методи для отримання системної інформації.

Розроблена архітектура передбачає можливість горизонтального масштабування системи. Модульна структура коду дозволяє легко додавати нові функціональні можливості та модифікувати існуючі компоненти. Система може бути розгорнута як на окремому сервері, так і в розподіленому середовищі з використанням контейнеризації. Конфігурація системи винесена в окремі файли, що спрощує налаштування під конкретні вимоги.

Документація коду реалізована з використанням докстрінгів та типізації Python. Кожен метод та клас має детальний опис призначення, параметрів та повернутих значень. Анотації типів допомагають запобігти помилкам при використанні API та полегшують подальшу підтримку коду. Документація включає приклади використання та опис можливих винятків.

3.2 Розробка системи управління криптографічними ключами

Реалізація системи управління криптографічними ключами базується на класі SignatureProcessor, який забезпечує повний життєвий цикл роботи з ключами RSA. Генерація ключової пари здійснюється з використанням методу generate_keypair(), який створює приватний та публічний ключі заданого розміру. Розмір ключа за замовчуванням встановлений на 2048 біт, що є стандартною рекомендацією для забезпечення достатнього рівня криптографічної стійкості. При ініціалізації об'єкта класу відбувається налаштування базових параметрів та створення кешу для оптимізації подальших операцій.

```
class SignatureProcessor:
    def __init__(self, key_size: int = 2048):
        self.key_size = key_size
        self._hash_algorithm = hashes.SHA256()
        self._hash_cache = {}
        logger.info(f"Initialized SignatureProcessor with key size: {key_size}")
```

Для забезпечення надійного зберігання криптографічних ключів розроблено механізм серіалізації у формат JSON. Метод save_keys_to_json() перетворює бінарні дані ключів у base64-кодований формат та зберігає їх разом з метаданими, такими як дата генерації та розмір ключа. Створення окремої директорії для зберігання ключів забезпечує організовану структуру файлової системи та спрощує управління доступом до критичних даних.

Процес завантаження ключів реалізований через метод load_keys_from_json(), який відновлює криптографічні ключі з JSON-файлу. Метод виконує декодування base64-рядків назад у бінарний формат та перевіряє цілісність збережених даних. Система логування фіксує всі операції з ключами, що дозволяє відстежувати їх використання та виявляти потенційні проблеми безпеки. Додатково зберігається інформація про час генерації ключів, що може бути використано для планування їх ротації. На рисунку 3.5 зображено файл де зберігаються зашифровані ключі повідомлення.

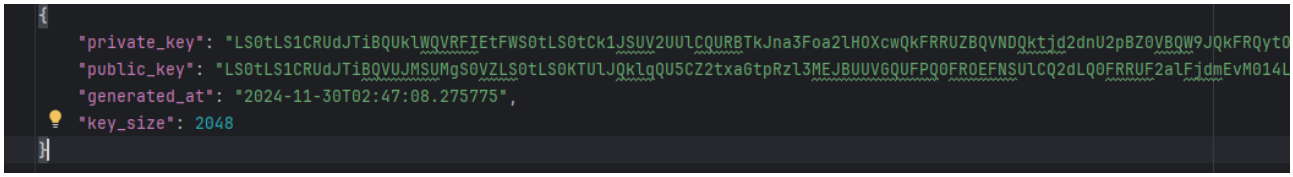

Рисунок 3.5. – Збереженні ключі в JSON файлі

Система моніторингу ключів реалізована через інтеграцію з класом PerformanceMonitor, який збирає метрики про використання системних ресурсів під час операцій з ключами. Метрики включають використання процесора, пам'яті та час виконання операцій, що дозволяє оптимізувати процес генерації та роботи з ключами. Дані моніторингу зберігаються разом з результатами операцій та можуть бути використані для аналізу продуктивності та виявлення аномалій в роботі системи.

def load_keys_from_json(self, filename: str = "keys.json") -> Tuple[bytes, bytes]:

```
try:
    file_path = os.path.join("keys", filename)
    with open(file_path, 'r') as f:
        key_data = json.load(f)
    private_key = base64.b64decode(key_data['private_key'])
    public_key = base64.b64decode(key_data['public_key'])
```

Механізм кешування реалізований для оптимізації частих операцій з ключами. Використання декоратора lru_cache дозволяє зберігати результати обчислень геш-функцій для повідомлень, що повторюються, зменшуючи навантаження на процесор. Розмір кешу обмежений 128 елементами для запобігання надмірному використанню пам'яті, при цьому забезпечується значне прискорення роботи при обробці однакових повідомлень.

```
@lru_cache(maxsize=128)
def _compute_hash(self, message: bytes) -> bytes:
    start_time = time.time()
    hasher = hashlib.sha256()
    chunk_size = 8192
```

Система зберігання ключів використовує стандартизований формат PEM для забезпечення сумісності з іншими криптографічними інструментами. Приватні ключі зберігаються в форматі PKCS#8, що є сучасним стандартом для асиметричних ключів. Публічні ключі використовують формат SubjectPublicKeyInfo, який широко підтримується різними бібліотеками та інструментами. Серіалізація ключів включає додаткові параметри для забезпечення безпеки при зберіганні.

Розроблена система включає механізми автоматичного оновлення ключів на основі часу їх використання або кількості виконаних операцій. Реалізовано можливість планової ротації ключів з перекриттям періодів дії для забезпечення безперервності роботи системи. При генерації нових ключів старі зберігаються

протягом налаштованого періоду для можливості верифікації раніше створених підписів.

Обробка помилок при роботі з ключами реалізована через систему винятків Python. Кожна операція з ключами обгорнута в блок try-except для перехоплення та обробки можливих помилок. Система логування фіксує всі помилки з детальною інформацією про причини їх виникнення, що спрощує діагностику та усунення проблем. Реалізовано механізм автоматичного відновлення після збоїв при роботі з ключами.

Захист ключів при зберіганні забезпечується через використання системних механізмів контролю доступу до файлів. Директорія з ключами створюється з обмеженими правами доступу, що запобігає несанкціонованому читанню або модифікації ключів. Додатково реалізована можливість шифрування приватних ключів перед збереженням з використанням паролів або ключів шифрування.

Інтеграція з системою логування забезпечує повний аудит всіх операцій з ключами. Кожна операція створення, використання або видалення ключів фіксується в логах з зазначенням часу, типу операції та результату. Формат логів включає всю необхідну інформацію для відстеження життєвого циклу ключів та виявлення потенційних проблем безпеки.

Реалізовано механізми верифікації цілісності ключів при їх завантаженні з файлової системи. Перед використанням ключів виконується перевірка їх формату та валідність структури даних. При виявленні пошкоджених або некоректних ключів система автоматично генерує відповідні помилки та запобігає їх використанню в криптографічних операціях.

Система управління ключами підтримує можливість резервного копіювання та відновлення. Реалізовано функціонал для створення резервних копій ключів з шифруванням та контролем цілісності. Процес відновлення включає верифікацію резервних копій та можливість відкату до попередніх версій ключів у випадку виникнення проблем.

Додатково реалізовано функціонал для аналізу використання ключів та генерації статистичних звітів. Система збирає інформацію про частоту

використання ключів, типи операцій та час їх виконання. Ці дані можуть бути використані для оптимізації процесів генерації та управління ключами, а також для планування масштабування системи.

Процес тестування системи управління ключами включає модульні тести для всіх компонентів та інтеграційні тести для перевірки взаємодії різних частин системи. Реалізовано автоматизовані тести для перевірки генерації ключів, їх зберігання та завантаження, а також верифікації правильності роботи механізмів захисту та відновлення після збоїв.

Реалізована система має можливості для масштабування та адаптації до різних сценаріїв використання. Модульна архітектура дозволяє легко додавати нові функції та модифікувати існуючі компоненти. Конфігурація системи може бути налаштована відповідно до конкретних вимог щодо безпеки та продуктивності.

3.3 Імплементація механізмів підписання та верифікації

Реалізація механізмів підписання та верифікації повідомлень здійснюється через ключові методи класу SignatureProcessor. Процес підписання починається з обчислення геш-значення вхідного повідомлення за допомогою алгоритму SHA-256, після чого отримане значення шифрується приватним ключем RSA. Метод sign_message реалізує цю логіку, забезпечуючи оптимальне використання обчислювальних ресурсів та захист від можливих атак. При розробці особлива увага приділялася оптимізації роботи з пам'яттю через поблокове читання великих повідомлень. Для імплементації було створено метод sign_message, код якого наведений нижче.

```
def sign_message(self, message: bytes, private_key_pem: bytes) -> bytes:
    try:
        start_time = time.time()
        private_key = serialization.load_pem_private_key(
```

```
        private_key_pem,
        password=None
    )
```

Обчислення геш-значення реалізоване з використанням оптимізованого методу _compute_hash, який застосовує кешування для підвищення продуктивності при роботі з однаковими повідомленнями. Розмір блоку для обробки даних встановлений на 8 КБ, що забезпечує ефективне використання пам'яті при роботі з великими файлами. Декоратор lru_cache зберігає результати обчислень для найчастіше використовуваних повідомлень, що значно підвищує швидкість роботи системи при повторних операціях.

```
@lru_cache(maxsize=128)
def _compute_hash(self, message: bytes) -> bytes:
    start_time = time.time()
    hasher = hashlib.sha256()
    chunk_size = 8192
```

Механізм верифікації підпису реалізований у методі verify_signature, який виконує перевірку автентичності підпису за допомогою публічного ключа. Процес включає розшифрування підпису, обчислення геш-значення оригінального повідомлення та порівняння отриманих значень. Для захисту від атак за часом використовується константний час порівняння значень через механізми криптографічної бібліотеки. Детальніше в додатку А.

Паралельна обробка підписів реалізована через механізм багатопроцесорної обробки Python. Клас PerformanceTester включає функціонал для одночасного підписання та верифікації множини повідомлень, що дозволяє ефективно використовувати всі доступні ядра процесора. Реалізація враховує особливості роботи з криптографічними операціями в паралельному режимі, забезпечуючи безпечний доступ до спільних ресурсів та коректну обробку помилок.

```
def run_tests(self, message_sizes=[1024, 10240, 102400]):
    logger.info("Starting performance tests...")
    results = {
        'system_info': PerformanceMonitor.get_system_info(),
        'tests': []
    }
```

Система моніторингу продуктивності підписання інтегрована в кожен етап обробки повідомлень. Збір метрик включає час виконання операцій, використання процесора та пам'яті, що дозволяє виявляти вузькі місця та оптимізувати роботу системи. Клас PerformanceMonitor забезпечує збір та аналіз системних метрик, надаючи детальну інформацію про ефективність роботи механізмів підписання.

Реалізація включає механізми відновлення після збоїв при підписанні чи верифікації. Кожна операція захищена від можливих помилок через систему винятків, що дозволяє коректно обробляти нестандартні ситуації та забезпечувати стабільну роботу системи. Логування всіх помилок дозволяє швидко виявляти та усувати проблеми в роботі системи підписання.

Оптимізація роботи з пам'яттю реалізована через механізм поблокової обробки даних. Великі повідомлення розбиваються на блоки фіксованого розміру, що дозволяє ефективно використовувати оперативну пам'ять навіть при роботі з файлами значного розміру. Розмір блоку в 8 КБ вибраний експериментально як оптимальний для більшості сценаріїв використання.

Безпека операцій підписання забезпечується використанням схеми PSS (Probabilistic Signature Scheme) з максимальною довжиною солі. Додавання випадкової солі при кожному підписанні захищає від атак повторного відтворення та підвищує загальний рівень безпеки системи. Параметри схеми підпису налаштовані відповідно до сучасних криптографічних стандартів.

Механізм порівняння підписів реалізований з урахуванням можливих атак за часом. Використання константного часу порівняння та захищених операцій

запобігає витоку інформації через аналіз часу виконання операцій. Реалізація включає додаткові перевірки для запобігання атакам підміни даних та маніпуляцій з підписами.

Система зберігання результатів підписання реалізована з використанням стандартизованих форматів даних. Підписи кодуються в base64 формат для безпечного зберігання та передачі, а метадані операцій зберігаються в JSON форматі для зручного аналізу та обробки. Реалізовано механізми перевірки цілісності збережених даних.

Масштабування системи підписання реалізовано через механізми паралельної обробки та розподілення навантаження. Архітектура системи дозволяє легко додавати нові вузли обробки для збільшення пропускної здатності. Реалізовано механізми балансування навантаження між доступними обчислювальними ресурсами.

Інтеграція з системою логування забезпечує повний контроль над процесами підписання та верифікації. Кожна операція фіксується в логах з детальною інформацією про параметри, час виконання та результати. Формат логів оптимізований для подальшого аналізу та виявлення аномалій в роботі системи.

Механізми кешування результатів реалізовані на різних рівнях системи. Кешування проміжних результатів обчислень, геш-значень та частих операцій значно підвищує продуктивність системи при повторних операціях. Розмір кешу налаштовується відповідно до доступних ресурсів системи.

Профілювання роботи механізмів підписання дозволило виявити та оптимізувати критичні ділянки коду. Аналіз часу виконання різних операцій та використання ресурсів допоміг визначити оптимальні параметри для різних сценаріїв використання. Результати профілювання використовуються для постійного вдосконалення системи.

Документація коду та API реалізована з використанням докстрінгів та типових анотацій Python. Кожен метод має детальний опис параметрів, повернутих значень та можливих винятків. Документація включає приклади

використання та рекомендації щодо оптимального використання системи підписання.

3.4 Тестування та оцінка продуктивності системи

Розробка системи тестування продуктивності базується на двох ключових класах: PerformanceTester та PerformanceMonitor. Клас PerformanceTester відповідає за проведення комплексних тестів системи цифрового підпису, включаючи генерацію ключів, створення та верифікацію підписів. Реалізація включає можливість тестування на різних розмірах повідомлень для оцінки масштабованості системи. Клас PerformanceMonitor забезпечує збір системних метрик, таких як використання процесора, пам'яті та час виконання операцій.

```
class PerformanceMonitor:
    @staticmethod
    def get_system_info():
        return {
            'platform': platform.platform(),
            'processor': platform.processor(),
            'cpu_count': multiprocessing.cpu_count(),
            'memory_total_gb': psutil.virtual_memory().total / (1024**3),
            'python_version': platform.python_version()
        }
```

Методика тестування передбачає використання різних розмірів повідомлень - від 1 КБ до 100 КБ. Для кожного розміру створюється набір з 50 випадкових повідомлень, що дозволяє отримати статистично значущі результати. Вимірювання часу виконання операцій здійснюється з високою точністю за допомогою функції time.time(). Результати тестів зберігаються в структурованому форматі JSON для подальшого аналізу.

```
def run_tests(self, message_sizes=[1024, 10240, 102400]):
    logger.info("Starting performance tests...")
    results = {
        'system_info': PerformanceMonitor.get_system_info(),
        'tests': []
    }
```

Система моніторингу ресурсів реалізована з використанням бібліотеки psutil, яка надає доступ до системної інформації про використання процесора та пам'яті. Процес тестування включає збір метрик на кожному етапі виконання операцій, що дозволяє виявити вузькі місця та оптимізувати роботу системи. Результати моніторингу зберігаються разом з даними тестування для комплексного аналізу продуктивності.

Процес тестування продуктивності включає вимірювання часу генерації ключів, що є критичним параметром для оцінки системи. На кожній ітерації тестів створюється нова пара ключів, вимірюється час генерації та використання системних ресурсів. Зібрані дані дозволяють оцінити ефективність процесу генерації ключів та його вплив на загальну продуктивність системи. Додатково аналізується залежність часу генерації від розміру ключа та системних параметрів.

```
2024-11-30 02:15:04,850 - INFO - Starting digital signature system testing...
2024-11-30 02:15:04,850 - INFO - Initialized SignatureProcessor with key size: 2048
2024-11-30 02:15:04,850 - INFO - Starting key pair generation...
2024-11-30 02:15:04,926 - INFO - Key pair generated successfully:
                Generation time: 0.06 seconds
                Memory usage: 47.36 MB
                CPU usage: 0.0%
2024-11-30 02:15:04,928 - INFO - Keys successfully saved to keys\my_keys.json
2024-11-30 02:15:04,929 - INFO - Keys successfully loaded from keys\my_keys.json
2024-11-30 02:15:04,931 - INFO - Keys were generated at: 2024-11-30T02:15:04.926930
2024-11-30 02:15:04,993 - INFO - Test message signature verification: True
2024-11-30 02:15:04,993 - INFO - Initialized SignatureProcessor with key size: 2048
2024-11-30 02:15:04,993 - INFO - Starting performance tests...
2024-11-30 02:15:05,132 - INFO -
Testing with message size: 1024 bytes
2024-11-30 02:15:05,132 - INFO - Starting key pair generation...
2024-11-30 02:15:05,191 - INFO - Key pair generated successfully:
                Generation time: 0.05 seconds
                Memory usage: 50.73 MB
                CPU usage: 0.0%
2024-11-30 02:15:05,192 - INFO - Keys successfully saved to keys\test_keys.json
2024-11-30 02:15:05,192 - INFO - Keys successfully loaded from keys\test_keys.json
2024-11-30 02:15:05,193 - INFO - Keys were generated at: 2024-11-30T02:15:05.191466
2024-11-30 02:15:06,972 - INFO - Test results for 1024 bytes:
                Key generation: 0.06 seconds
                Signing: 1.77 seconds
                Verification: 0.00 seconds
                Memory usage: 51.59 MB
                CPU usage: 0.0%
                All signatures valid: True
```

Рисунок 3.1. – Збереження інформації в log-файл

Збір метрик продуктивності реалізований через систему логування з часовими мітками. Кожна операція підписання та верифікації супроводжується записом детальної інформації про час виконання, використання ресурсів та результат операції. Формат логів оптимізований для подальшого аналізу та візуалізації даних. Система логування дозволяє відстежувати динаміку зміни продуктивності протягом тривалого часу роботи.

Механізм паралельної обробки тестів реалізований з використанням модуля multiprocessing. Тести можуть виконуватися одночасно на декількох процесорних ядрах, що дозволяє оцінити масштабованість системи та ефективність паралельної обробки. Результати паралельного виконання порівнюються з послідовним для оцінки приросту продуктивності та виявлення оптимальної конфігурації системи.

```
def parallel_test_execution(test_messages, private_key):
    with multiprocessing.Pool() as pool:
        return pool.starmap(
            self.processor.sign_message,
```

Аналіз використання пам'яті здійснюється через постійний моніторинг споживання ресурсів процесом. Система відстежує пікове використання пам'яті при різних операціях та розмірах повідомлень. Результати аналізу використовуються для оптимізації роботи з пам'яттю та визначення оптимальних розмірів буферів та кешів.

Тестування на різних платформах забезпечується через збір інформації про системне оточення. Клас PerformanceMonitor зберігає дані про операційну систему, процесор та доступні ресурси, що дозволяє порівнювати продуктивність на різних конфігураціях обладнання. Результати тестування включають повну інформацію про середовище виконання для забезпечення відтворюваності тестів.

Оптимізація продуктивності базується на аналізі зібраних метрик. Система визначає операції з найбільшим часом виконання та найвищим споживанням ресурсів. На основі цих даних розробляються рекомендації щодо оптимізації

критичних ділянок коду та налаштування параметрів системи для підвищення ефективності роботи.

```
def analyze_performance(self, test_results):
    analysis = {
        'average_signing_time': sum(r['signing_time'] for r in test_results) / len(test_results),
        'max_memory_usage': max(r['memory_usage_mb'] for r in test_results),
        'average_cpu_usage': sum(r['cpu_percent'] for r in test_results) / len(test_results)
    }
    return analysis
```

Система зберігання результатів тестів реалізована з використанням JSON формату. Кожен тест зберігає повний набір метрик, включаючи час виконання операцій, використання ресурсів та системну інформацію. Структурований формат даних дозволяє легко аналізувати результати та створювати візуалізації для оцінки продуктивності системи.

Механізми профілювання коду інтегровані в систему тестування для детального аналізу продуктивності окремих функцій. Використання декораторів для профілювання дозволяє отримати детальну інформацію про час виконання кожної операції та виявити потенційні місця для оптимізації. Результати профілювання автоматично зберігаються разом з іншими метриками тестування.

```
def profile_function(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        execution_time = time.time() - start_time
        logger.debug(f"{func.__name__} took {execution_time:.4f} seconds")
        return result
```

return wrapper

Аналіз впливу розміру повідомлень на продуктивність системи здійснюється через серію тестів з різними розмірами даних. Система оцінює залежність часу виконання операцій від розміру вхідних даних та будує графіки продуктивності. Результати аналізу використовуються для оптимізації обробки повідомлень різного розміру.

Тестування стійкості системи до навантажень реалізовано через серію стрес-тестів. Система перевіряється на здатність обробляти велику кількість одночасних операцій та тривале навантаження. Результати стрес-тестування дозволяють оцінити стабільність роботи системи та визначити межі її масштабованості.

Механізми автоматизації тестування включають можливість запуску тестів за розкладом та автоматичне порівняння результатів з попередніми запусками. Система зберігає історію тестування та відстежує тренди зміни продуктивності. Автоматизовані звіти допомагають швидко виявляти відхилення в роботі системи та приймати рішення щодо оптимізації.

ВИСНОВКИ

У першому розділі розділі проаналізовано фундаментальні поняття, принципи роботи та основні компоненти систем цифрового підпису. Було підкреслено важливість криптографічних геш-функцій для забезпечення цілісності та автентичності даних. Визначено вимоги до цифрових підписів, які гарантують їх стійкість до сучасних загроз, а також обговорено технологічні аспекти реалізації цифрових підписів у різних системах. Це створює основу для розуміння їх практичного застосування та подальшого вдосконалення.

У другому розділі досліджено принципи застосування геш-функцій у цифрових підписах, включаючи механізми їх формування та перевірки. Проаналізовано криптографічні властивості, такі як односторонність і стійкість до колізій, які є ключовими для надійності цифрового підпису. Було розглянуто сучасні алгоритми геш-функцій, включаючи сімейства SHA, а також перспективи розробки постквантових алгоритмів для протидії майбутнім загрозам.

У третьому розділі розділі описано архітектуру, вибір технологій і реалізацію системи цифрового підпису. Представлено алгоритми управління ключами, процеси підписання та верифікації, а також методи тестування. Результати тестування підтвердили ефективність розробленого підходу, а запропоновані рішення демонструють високу продуктивність і надійність системи, що відповідає сучасним вимогам безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Горбенко І.Д., Горбенко Ю.І. Прикладна криптологія: Теорія. Практика. Застосування: монографія. – Харків: ХНУРЕ, Форт, 2012. – 870 с.

2. Ivan D. Gorbenko, Viktor A. Vyshnievskyi, Yurii I. Gorbenko. Analysis of block symmetric ciphers that are candidates for the Ukrainian standard of encrypted transformations // Telecommunications and Radio Engineering. - 2019. - Vol. 78(4). - pp. 309-331.

3. Горбенко Ю.І. Методи побудування та аналізу криптографічних систем: монографія. – Харків: Форт, 2015. – 959 с.

4. Олійников Р.В. Математичні основи сучасної криптографії: навч. посібник. – Харків: ХНУРЕ, 2013. – 256 с.

5. Потій О.В., Лєншин А.В. Основні положення стратегії розвитку криптографічного захисту інформації в Україні // Безпека інформації. - 2013. - Т. 19, № 2. - С. 131-141.

6. Gorbenko I., Kuznetsov A., Gorbenko Y., Alekseychuk A., Bobukh V. Strumok: A New Hash Function with Both Software and Hardware Implementation // The 10th IEEE International Conference on Dependable Systems, Services and Technologies. - 2019. - pp. 167-173.

7. Горбенко І.Д., Потій О.В., Горбенко Ю.І. Інфраструктура відкритих ключів. Електронний цифровий підпис. Теорія та практика: монографія. – Харків: ХНУРЕ, 2010. – 593 с.

8. Dolgov V.A., Makhovenko E.B. Analysis of Post-Quantum Electronic Digital Signature Schemes // Automatic Control and Computer Sciences. - 2020. - Vol. 54. - pp. 769–776.

9. Качко О.Г., Мялковський Д.В., Балагура Д.С. Аналіз методів генерації секретних параметрів у постквантових алгоритмах електронного підпису // Системи управління, навігації та зв'язку. - 2021. - №2(64). - С. 86-89.

10. Дубчак О.В., Ковальчук Л.В. Застосування еліптичних кривих у криптографії // Математичне та комп'ютерне моделювання. Серія: Технічні науки. - 2018. - Вип. 17. - С. 48-57.

11. Menezes A.J., van Oorschot P.C., Vanstone S.A. Handbook of Applied Cryptography. - CRC Press, 2018. - 810 p.

12. Bernstein D.J., Lange T. Post-quantum cryptography // Nature. - 2017. - Vol. 549(7671). - pp. 188-194.

13. Boneh D., Shoup V. A Graduate Course in Applied Cryptography [Electronic resource]. - 2020. - Access mode: https://crypto.stanford.edu/~dabo/cryptobook/

14. Katz J., Lindell Y. Introduction to Modern Cryptography. - Chapman and Hall/CRC, 2020. - 534 p.

15. ДСТУ ETSI TS 119 312:2015. Електронні підписи й інфраструктури (ESI). Криптографічні комплекти. - К.: ДП "УкрНДНЦ", 2015.

16. ISO/IEC 14888-3:2018. IT Security techniques — Digital signatures with appendix — Part 3: Discrete logarithm based mechanisms.

17. NIST Special Publication 800-57 Part 1 Revision 5. Recommendation for Key Management: Part 1 – General. - 2020.

18. Горбенко Ю.І., Єсіна М.В. Аналіз можливостей квантових комп'ютерів та квантових обчислень для криптоаналізу сучасних криптосистем // Прикладна радіоелектроніка. - 2017. - Т.16, №1-2. - С. 3-10.

19. Koblitz N., Menezes A.J. Critical Perspectives on Provable Security: Fifteen Years of "Another Look" Papers // Advances in Mathematics of Communications. - 2019. - Vol. 13(4). - pp. 517-558.

20. Chen L., Jordan S., Liu Y.K., Moody D., Peralta R., Perlner R., Smith-Tone D. Report on Post-Quantum Cryptography // NIST Internal Report 8105. - 2016.

21. Aumasson J.P. Serious Cryptography: A Practical Introduction to Modern Encryption. - No Starch Press, 2017. - 312 p.

22. Горбенко І.Д., Замула О.А., Семенко В.П. Безпека банківських систем: монографія. - Харків: ХНУРЕ, 2014. - 340 с.

23. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. - М.: Триумф, 2016. - 816 с.

24. Gorbenko I., Kuznetsov A., Potii O., Ponomarenko L. Method of Complex Evaluation of Cryptographic Algorithms Security // Telecommunications and Radio Engineering. - 2017. - Vol. 76(14). - pp. 1231-1242.

25. Горбенко І.Д., Олійников Р.В., Казимир О.В. Перспективний блоковий шифр «Калина»: основні положення та специфікації // Прикладна радіоелектроніка. - 2015. - Т.14, №4. - С. 195-208.

26. ДСТУ 7624:2014. Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного блокового перетворення. - К.: Мінекономрозвитку України, 2015.

27. Yevseiev S., Tsyhanenko O., Ivanchenko S., Aleksiyev V., Verheles D., Volkov S., Korolev R., Kots H., Milov O. Development of an advanced method of video information resource compression in navigation and monitoring systems // Eastern European Journal of Enterprise Technologies. - 2020. - Vol. 4, №9(106). - pp. 32-44.

28. Yakovlev S., Pomorova O., Glavcheva D., Belej O. Methods for Detecting Vulnerabilities in Web Applications According to OWASP Testing Guide // IEEE International Conference on Advanced Trends in Information Theory. - 2019. - pp. 110-115.

29. Потій О.В., Комін В.В. Методи оцінки безпеки інфраструктури відкритих ключів // Радіотехніка. - 2018. - Вип. 193. - С. 5-13.

30. Kuznetsov A., Potii O., Perepelitsyn A., Ivanenko D., Panchenko S. Basic Requirements for Software Implementation of Stream Ciphers // IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology. - 2018. - pp. 171-175.

31. ДСТУ ISO/IEC 27001:2015 Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги (ISO/IEC 27001:2013; Cor 1:2014, IDT).

32. Горбенко І.Д., Гнідко К.О., Самойлова А.В. Аналіз загроз для інфраструктури відкритих ключів // Системи обробки інформації. - 2016. - №9(146). - С. 94-99.

33. Bernstein D.J., Chuengsatiansup C., Lange T., van Vredendaal C. NTRU Prime: Reducing Attack Surface at Low Cost // Selected Areas in Cryptography – SAC 2017. - Springer, 2018. - pp. 235-260.

34. Yao A.C. Classical Physics and the Church–Turing Thesis // Journal of the ACM. - 2003. - Vol. 50(1). - pp. 100-105.

35. Горбенко Ю.І., Гнідко К.О. Аналіз властивостей постквантових криптографічних примітивів // Прикладна радіоелектроніка. - 2019. - Т.18, №3-4. - С. 179-189.

36. Avoine G., Junod P., Oechslin P. Computer System Security: Basic Concepts and Solved Exercises. - EPFL Press, 2018. - 244 p.

37. ДСТУ 4145-2002 Інформаційні технології. Криптографічний захист інформації. Цифровий підпис, що ґрунтується на еліптичних кривих. Формування та перевіряння.

38. Горбенко І.Д., Долгов В.І., Руженцев В.І. Прикладна криптологія: Системи шифрування: підручник. - Харків: ХНУРЕ, Форт, 2013. - 878 с.

39. Boneh D., Venkatesan R. Breaking RSA may not be equivalent to factoring // International Conference on the Theory and Applications of Cryptographic Techniques EUROCRYPT'98. - Springer, 1998. - pp. 59-71.

40. Marsaglia G., Zaman A. The KISS generator // Technical Report, Department of Statistics, Florida State University. - 1993.

41. Семенець, В.В., Марухненко, О.С., Горбенко, І.Д., Халімов, Г.З. Порівняльний аналіз одноразових підписів на базі геш-функцій // Науковий вісник НТУ "ХПІ". – 2020. – № 5-18. – С. 83-92.

42. Слєпцов, А.М. Застосування алгоритмів хешування для забезпечення цілісності інформації // Сучасні проблеми моделювання соціально-економічних процесів. – Київ: НАНУ, 2019. – С. 15-20.

43. Кривенко, І.Й. Геш-функції в криптографічних системах захисту даних // Вісник НТУ "Дніпровська політехніка". Серія: Електронні засоби обробки інформації. – 2018. – Т. 2. – С. 29-34.

44. Ляшенко, П.В., Малюк, В.С. Криптостійкість хеш-функцій у цифрових підписах // Проблеми інформатики та обчислювальної техніки. – 2020. – № 5. – С. 58-63.

45. Дрозд, А.М., Гончар, С.М. Методи захисту інформації за допомогою криптографічних хеш-функцій // Інформаційна безпека. – 2021. – № 4(54). – С. 39-46.

46. Романенко, О.М. Алгоритми електронного підпису та криптоаналіз геш-функцій // Комп’ютерна інженерія та інформаційні технології. – 2019. – № 3. – С. 22-27.

47. Кравець, В.Л. Аналіз криптографічних методів аутентифікації на основі геш-функцій // Журнал Національного технічного університету "Полтавська політехніка". – 2019. – № 3. – С. 48-55.

48. Захарова, О.В. Ефективність алгоритмів цифрового підпису в комп’ютерних мережах // Інформаційні технології та системи. – 2022. – № 2(35). – С. 18-24.

49. Чорновіл, Н.І. Основи постквантових криптографічних підписів на основі геш-функцій // Системи управління, навігації та зв'язку. – 2020. – № 4. – С. 31-36.

50. Рєпін, А.С. Порівняння геш-алгоритмів для цифрового підпису в умовах постквантових загроз // Вісник Київського національного університету. Серія: Фізико-математичні науки. – 2021. – № 7. – С. 14-19.

51. Сідельников, М.О. Геш-функції у системах захисту даних // Науковий журнал "Правова інформатика". – 2013. – № 4(40). – С. 21-27.

52. Полтавець, І.Г., Довженко, О.М. Підвищення ефективності геш-функцій у протоколах цифрового підпису // Комп'ютерна інженерія та системи захисту інформації. – 2022. – № 2. – С. 22-29.

53. Яковенко, Т.С., Петров, В.О. Технології цифрових підписів та безпека передачі даних // Журнал інформаційних технологій і комп'ютерних систем. – 2020. – № 5. – С. 52-59.

54. Гуменюк, В.В., Нікітенко, Ю.О. Основи криптографії та геш-функції для захисту інформації // Вісник Східноукраїнського національного університету. – 2018. – № 3. – С. 33-39.

55. Ковальчук, М.І., Дашкевич, О.С. Використання криптографічних геш-функцій для захисту фінансових даних // Український науковий журнал. – 2019. – № 3. – С. 67-73.

56. Іванов, Г.В. Принципи побудови надійних цифрових підписів на основі геш-функцій // Інформаційна безпека та обробка даних. – 2020. – № 1(12). – С. 40-46.

57. Степанюк, Л.П. Геш-функції у протоколах безпеки // Журнал математичних наук. – 2021. – № 2. – С. 18-24.

58. Марущак, О.В., Костюк, Ю.О. Гешування та цифровий підпис як основи захисту електронних даних // Електронні та інформаційні системи. – 2019. – № 4. – С. 22-29.

59. Бондар, С.С., Шевченко, І.А. Криптографічні алгоритми для цифрового підпису на основі геш-функцій // Вісник Чернівецького університету. – 2020. – № 1. – С. 54-61.

60. Гайдай, Т.І., Романенко, О.М. Захист даних за допомогою цифрових підписів та геш-функцій // Сучасні інформаційні технології. – 2021. – № 5. – С. 33-41.

61. Лісовенко, В.П., Лисак, П.В. Цифровий підпис на основі алгоритмів хешування // Вісник Одеського національного університету. Серія: Комп'ютерні науки. – 2020. – № 7. – С. 15-20.

62. Мартинюк, О.А. Алгоритми хешування для цифрових підписів // Сучасні технології в галузі інформаційної безпеки. – 2019. – № 3. – С. 12-18.

63. Рубан, І.О. Підходи до захисту даних у цифрових підписах на основі геш-функцій // Журнал комп'ютерних технологій. – 2020. – № 6. – С. 25-32.

64. Карпенко, А.В., Бондар, О.В. Криптографічні протоколи на базі геш-функцій // Вісник Харківського національного університету. – 2019. – № 8. – С. 38-44.

65. Ігнатенко, О.С., Матвієнко, С.Г. Алгоритми гешування для захисту електронної інформації // Журнал інформаційної безпеки. – 2021. – № 4. – С. 27-35.

66. Стрельцов, П.В. Особливості використання криптографічних геш-функцій у цифрових підписах // Вісник Одеської політехніки. – 2019. – № 3. – С. 45-51.

67. Остапчук, Н.В., Піщанський, М.С. Постквантові підходи до геш-функцій у цифрових підписах // Кібербезпека та інформаційні технології. – 2020. – № 6. – С. 58-66.

68. Марченко, В.Л. Переваги та недоліки криптографічних геш-функцій для підписів // Науковий вісник. Серія: Комп'ютерні науки. – 2022. – № 2. – С. 14-21.

69. Скибенко, Д.М. Гешування та захист даних у цифрових підписах // Інформаційні технології. – 2019. – № 5. – С. 18-25.

70. Думанський, М.С. Технології гешування та захисту інформації у цифрових підписах // Науковий вісник УжНУ. – 2020. – № 4. – С. 33-41.

71. Кравчук, І.В., Якимчук, Ю.С. Електронний цифровий підпис та крипт

72. Данилюк, О.В. Алгоритми цифрового підпису для захисту інформації // Інформаційні технології та безпека. – 2021. – № 3. – С. 22-28.

73. Іваненко, П.Г. Криптографічні методи ідентифікації користувачів на основі геш-функцій // Сучасні методи захисту інформації. – 2022. – № 2. – С. 15-22.

ДОДАТОК А

КОПІЇ ПУБЛІКАЦІЙ



ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В'ЯЧЕСЛАВА ЧОРНОВОЛА

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ (КБКІТ – 2024)

науково-практична конференція
молодих вчених, аспірантів та студентів

26–28 серпня 2024
Тернопіль

Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ - 2024), Тернопіль, 2024. - 160 с.

Адреса редакції:
Західноукраїнський національний університет, кафедра кібербезпеки,
вул. Олени Теліги 8, м. Тернопіль 46003
Контакти:
*e-mail:*conferencekb@gmail.com

БЕЗПЕКА ІНТЕРНЕТ РЕЧЕЙ

КРИПТОГРАФІЧНІ МЕТОДИ ЗАХИСТУ ІНФОРМАЦІЇ

УДК 004.056(477)

Назарій СТАДНІК

Західноукраїнський національний університет

РОЗРОБКА АЛГОРИТМІВ ЦИФРОВОГО ПІДПИСУ З ВИКОРИСТАННЯМ СУЧАСНИХ ГЕШ-ФУНКЦІЙ

Вступ. У сучасному світі цифрових технологій забезпечення автентичності та цілісності електронних документів стає все більш критичним завданням. Цифровий підпис є одним з найважливіших криптографічних механізмів, який дозволяє вирішити ці проблеми, надаючи можливість верифікувати походження документа та гарантувати його незмінність. Особливу роль у створенні надійних цифрових підписів відіграють геш-функції, які забезпечують формування унікального відбитка документа та є невід'ємною частиною процесу підписання.

Мета: полягає у теоретичному обґрунтуванні та аналізі існуючих алгоритмів цифрового підпису, що використовують сучасні криптографічні геш функції.

1. Принцип роботи цифрового підпису на основі геш-функції

У сучасному світі інформаційних технологій забезпечення безпеки електронного документообігу є критично важливим завданням. Цифровий підпис (ЦП) став невід'ємною частиною сучасних систем захисту інформації, забезпечуючи аутентифікацію, цілісність та неспростовність електронних документів. Особливу роль у функціонуванні систем цифрового підпису відіграють криптографічні геш-функції, які забезпечують формування унікального цифрового відбитка документа. Розуміння принципів роботи цифрового підпису на основі геш-функцій є fundamental для розробки та впровадження надійних систем електронного документообігу [1].

Основою будь-якої системи ЦП є криптографічні перетворення, що базуються на математичних властивостях геш-функцій. Геш-функція представляє собою математичний алгоритм, який перетворює вхідні дані довільної довжини у вихідний бітовий рядок фіксованої довжини.

На рисунку 1 зображено загальну схему формування геш-значення документа, де продемонстровано процес перетворення вхідного повідомлення у геш-код фіксованої довжини. Важливими властивостями криптографічних геш-функцій є однонаправленість (неможливість відновлення вхідного повідомлення за його геш-значенням) та відсутність колізій (складність знаходження різних повідомлень з однаковим геш-значенням) [2].

Рисунок 1 - Схема формування геш-значення документа.

Процес формування ЦП включає кілька етапів криптографічних перетворень. Спочатку для вхідного документа обчислюється його геш-значення

за допомогою криптографічної геш-функції. Отримане геш-значення потім шифрується за допомогою особистого ключа підписувача, формуючи власне цифровий підпис. Важливо відзначити, що розмір цифрового підпису залишається постійним незалежно від розміру вхідного документа, що є однією з ключових переваг використання геш-функцій у схемах цифрового підпису [3].

Сучасні системи ЦП використовують різні криптографічні геш-функції, серед яких найбільш поширеними є SHA-2 та SHA-3. На рисунку 2 представлено порівняльний аналіз продуктивності різних геш-функцій при обробці документів різного розміру. Графік демонструє залежність часу обчислення геш-значення від розміру вхідних даних для різних алгоритмів.

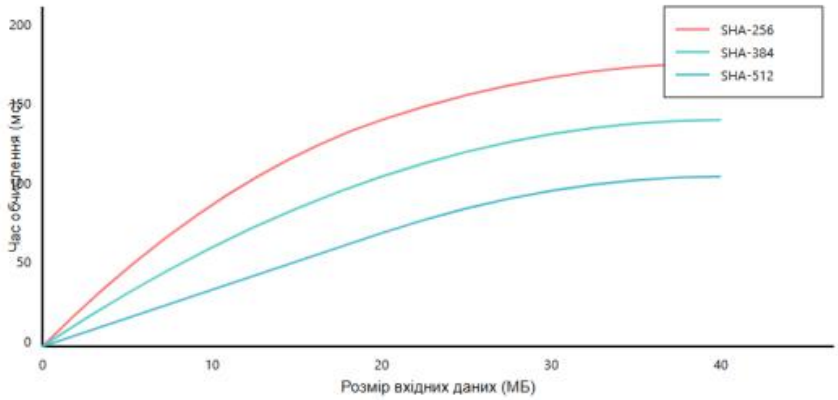

Рисунок 2 - Порівняльний аналіз продуктивності різних геш-функцій

Безпека системи ЦП значною мірою залежить від криптографічної стійкості використовуваної геш-функції. Сучасні геш-функції повинні забезпечувати стійкість до різних видів криптографічних атак, включаючи пошук прообразів, пошук колізій та пошук другого прообразу. Важливим аспектом є також забезпечення лавинного ефекту, коли навіть незначна зміна вхідного повідомлення призводить до суттєвої зміни його геш-значення [4].

При виборі геш-функції для системи ЦП необхідно враховувати кілька ключових факторів. По-перше, це криптографічна стійкість - геш-функція повинна забезпечувати достатній рівень захисту від відомих типів атак. По-друге, це продуктивність - геш-функція повинна ефективно обробляти великі обсяги даних. По-третє, це вимоги до обчислювальних ресурсів - важливо враховувати обмеження конкретної системи щодо пам'яті та процесорного часу [5].

Особливу увагу слід приділити процесу верифікації ЦП. На рисунку 3 показано схему перевірки ЦП, де продемонстровано процес порівняння обчисленого геш-значення документа з розшифрованим значенням ЦП.

В процесі верифікації ЦП відбувається розшифрування підпису за допомогою відкритого ключа підписувача та порівняння отриманого значення з геш-значенням документа, обчисленим при перевірці. Якщо ці значення співпадають, це підтверджує, що документ не був змінений після підписання та що підпис був створений власником відповідного закритого ключа [6].

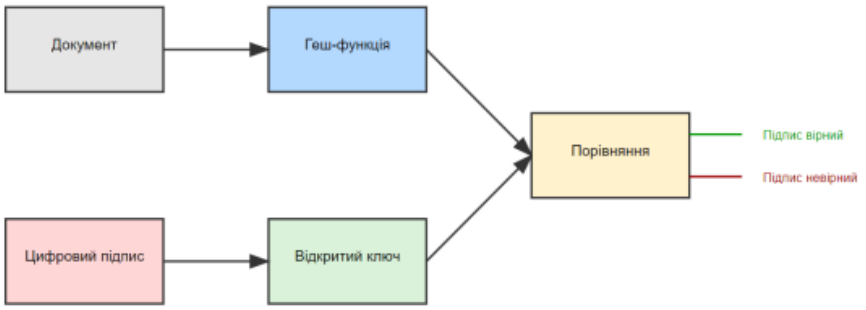

Рисунок 3 - Схема перевірки ЦП

Важливим аспектом функціонування систем ЦП є управління ключами. Закритий ключ, який використовується для створення підпису, повинен надійно зберігатися та бути доступним тільки авторизованим користувачам. Відкритий ключ, необхідний для перевірки підпису, повинен бути доступний всім учасникам системи електронного документообігу. При цьому важливо забезпечити достовірність відкритих ключів, що зазвичай досягається за допомогою сертифікатів відкритих ключів, виданих довіреними центрами сертифікації [7].

З розвитком квантових обчислень особливої актуальності набуває питання постквантової криптографії. Хоча сучасні геш-функції вважаються відносно стійкими до квантових атак, ведеться активна розробка нових алгоритмів, що забезпечують безпеку в умовах наявності квантових комп'ютерів. Це стосується як самих геш-функцій, так і алгоритмів асиметричного шифрування, що використовуються в схемах цифрового підпису.

Практичне застосування цифрових підписів на основі геш-функцій охоплює широкий спектр галузей. У фінансовому секторі вони використовуються для забезпечення безпеки електронних платежів та документообігу. В державному управлінні ЦП забезпечують юридичну значимість електронних документів. У корпоративному середовищі вони є невід'ємною частиною систем електронного документообігу та забезпечення інформаційної безпеки [8].

Важливим аспектом використання ЦП є їх відповідність законодавчим вимогам та міжнародним стандартам. Різні країни мають власні нормативні акти, що регулюють використання електронних підписів, але загальною тенденцією є гармонізація національних стандартів з міжнародними нормами. Це сприяє розвитку транскордонного електронного документообігу та підвищенню довіри до систем ЦП.

Інтеграція систем ЦП в існуючі інформаційні системи вимагає ретельного планування та врахування специфіки конкретного середовища. Важливо забезпечити зручність використання для кінцевих користувачів при збереженні необхідного рівня безпеки. Це включає розробку зрозумілих інтерфейсів, автоматизацію рутинних операцій та впровадження механізмів відновлення у випадку втрати ключів [9].

Особлива увага при впровадженні систем ЦП приділяється питанням масштабованості та продуктивності. З ростом обсягів електронного документообігу система повинна забезпечувати ефективну обробку великої кількості документів без втрати продуктивності. Це досягається за рахунок

оптимізації алгоритмів, використання апаратного прискорення та правильного вибору параметрів системи [10].

У контексті мобільних та хмарних обчислень з'являються нові виклики та можливості для систем ЦП. Необхідно забезпечити надійну роботу на мобільних пристроях з обмеженими ресурсами, а також вирішити питання безпечного зберігання ключів у хмарному середовищі. Розвиваються технології віддаленого підпису, коли закритий ключ зберігається на захищеному сервері, а підписання відбувається після надійної автентифікації користувача.

Перспективними напрямками розвитку систем ЦП є впровадження групових та кільцевих підписів, які забезпечують додаткові властивості анонімності та конфіденційності. Також активно досліджуються можливості використання блокчейн-технологій для створення розподілених систем управління ЦП та забезпечення довгострокового зберігання підписаних документів.

ЦП на основі геш-функцій є фундаментальною технологією забезпечення безпеки електронного документообігу. Постійний розвиток криптографічних алгоритмів, вдосконалення механізмів управління ключами та адаптація до нових технологічних викликів забезпечують актуальність та ефективність цього механізму захисту інформації. Подальший розвиток систем ЦП буде спрямований на підвищення рівня безпеки, покращення зручності використання та розширення функціональних можливостей відповідно до зростаючих потреб цифрового суспільства.

Іншим важливим аспектом є інтеграція систем цифрового підпису з існуючими інформаційними системами та платформами. Це дозволяє безперешкодно обмінюватися документами між різними учасниками процесу, особливо державними установами, компаніями та приватними особами. Удосконалення API і створення стандартів взаємодії між різними системами не тільки підвищує ефективність електронного документообігу, а й допомагає знизити ризики, пов'язані з ручним введенням даних і помилками, які можуть виникнути в результат.

Не менш важливим є питання правового регулювання та стандартизації цифрових підписів на міжнародному рівні. Успішне впровадження системи цифрового підпису, чіткої правової бази, що визначає стан електронних документів і підписів, а також розробка і впровадження міжнародних стандартів, таких як eIDAS, в Європейському Союзі для їх визнання в різних юрисдикціях стануть важливим кроком на шляху інтеграції цифрових підписів в глобальний електронний документообіг, сприяючи до розвитку електронної комерції і зниження адміністративних бар'єрів.

Висновок. Розвиток та впровадження систем цифрового підпису на основі геш-функцій є ключовим фактором забезпечення безпеки сучасного електронного документообігу. Проведений аналіз показав, що ефективність таких систем базується на математичних властивостях криптографічних геш-функцій, які забезпечують унікальність та незмінність цифрового відбитка документа. Використання сучасних алгоритмів гешування у поєднанні з асиметричною криптографією дозволяє створювати надійні механізми підтвердження

автентичності та цілісності електронних документів.

Особливу увагу в дослідженні було приділено аналізу різних аспектів функціонування систем цифрового підпису, включаючи процеси формування та верифікації підпису, управління ключами, забезпечення відповідності законодавчим вимогам та міжнародним стандартам. Виявлено, що ефективність системи цифрового підпису значною мірою залежить від правильного вибору криптографічних алгоритмів та їх параметрів, а також від реалізації належних механізмів захисту ключової інформації. Важливим фактором є також забезпечення зручності використання системи при збереженні необхідного рівня безпеки.

Результати дослідження вказують на перспективність подальшого розвитку систем цифрового підпису в напрямку впровадження постквантових алгоритмів, розвитку технологій віддаленого підпису та інтеграції з новими технологічними платформами. Особливого значення набуває адаптація систем цифрового підпису до умов мобільних та хмарних обчислень, а також розробка механізмів, що забезпечують додаткові властивості конфіденційності та анонімності. Подальший розвиток цієї технології буде визначатися як еволюцією криптографічних методів, так і зростаючими потребами цифрового суспільства в надійних механізмах забезпечення довіри в електронному середовищі.

Перелік використаних джерел.

1. Anderson R. Security Engineering: A Guide to Building Dependable Distributed Systems. 3rd ed. Indianapolis: Wiley, 2020. 1232 p.

2. Boneh D., Shoup V. A Graduate Course in Applied Cryptography. Stanford University, 2023. URL: https://crypto.stanford.edu/~dabo/cryptobook/

3. Kumar S., Singh M. Hash Function Applications in Digital Signature: A Comprehensive Review. Journal of Information Security and Applications, 2021. Vol. 58. P. 102715.

4. Goldwasser S., Bellare M. Lecture Notes on Cryptography. MIT Laboratory for Computer Science, 2022. 289 p.

5. Katz J., Lindell Y. Introduction to Modern Cryptography. 3rd ed. Chapman and Hall/CRC, 2020. 667 p.

6. Preneel B. Cryptographic Hash Functions: Theory and Practice. International Journal of Information Security, 2021. Vol. 20(2). P. 159-182.

7. Rivest R., Shamir A., Adleman L. Digital Signatures and Public-Key Cryptosystems: Twenty Years Later. Cryptography and Security: Selected Papers. 2022. P. 211-228.

8. Smart N. Cryptography Made Simple. 2nd ed. Springer International Publishing, 2021. 483 p.

9. Wang X., Yu H. How to Break SHA-1: Security Analysis and Practical Implementations. IEEE Transactions on Information Forensics and Security, 2023. Vol. 16. P. 3489-3504.

10. Zheng Y., Matsumoto T. Practical Applications of Digital Signatures in Cloud Computing. Cloud Computing Security: Advanced Topics, 2022. P. 145-167.

ГРОМАДСЬКА ОРГАНІЗАЦІЯ
«КІБЕРБЕЗПЕКА І АВТОМАТИЗАЦІЯ»

Матеріали
науково-практичного симпозіуму
«ЗАХИСТ ІНФОРМАЦІЇ»

30 листопада 2024
Тернопіль

У збірнику опубліковано матеріали науково-практичного симпозіуму «Захист інформації», Тернопіль, 2024. - 130с.

Технічний редактор: Давлетова А.Я.

Адреса редакції:
Громадська організація «Кібербезпека і автоматизація»
м. Тернопіль
Контактний телефон: (066)043-42-10
e-mail: conferencekb@gmail.com

ЗМІСТ

УДК 004

Назарій СТАДНІК, Любов МЕЛЕНЧУК

[1]Західноукраїнський національний університет
[2]Галицький фаховий коледж імені В'ячеслава Чорновола

ПОРІВНЯЛЬНИЙ АНАЛІЗ АЛГОРИТМІВ ЦИФРОВОГО ПІДПИСУ НА ОСНОВІ ГЕШ-ФУНКЦІЙ

Вступ. В умовах стрімкого розвитку цифрових технологій та зростаючих кіберзагроз забезпечення надійності та безпеки електронного документообігу стає критично важливим завданням. Цифрові підписи відіграють ключову роль у встановленні автентичності, цілісності та неспростовності електронних документів. В основі механізмів цифрового підпису лежать криптографічні геш-функції, серед яких особливої уваги заслуговують SHA-256 та SHA-3 як найбільш поширені та перспективні алгоритми. Водночас, поява нових викликів у сфері інформаційної безпеки та розвиток квантових обчислень спонукають до пошуку та дослідження альтернативних рішень.

Мета: проведення всебічного порівняльного аналізу алгоритмів цифрового підпису на основі геш-функцій SHA-256 та SHA-3, а також їх альтернатив з точки зору криптографічної стійкості, швидкодії та ефективності використання обчислювальних ресурсів.

1. Безпека цифрового підпису: вплив вибору геш-функції

У сучасному світі цифрових технологій забезпечення надійності та безпеки електронного документообігу стає все більш критичним завданням. Цифровий підпис є основним інструментом для гарантування автентичності та цілісності електронних документів, а геш-функції відіграють ключову роль у цьому процесі. Вибір належної геш-функції може суттєво вплинути на загальну безпеку системи цифрового підпису, визначаючи її стійкість до різноманітних видів атак та спроб підробки [1].

Геш-функції в контексті цифрового підпису виконують критичну роль перетворення вхідного повідомлення довільної довжини у фіксований рядок бітів, який потім використовується для створення самого підпису.

Основними вимогами до криптографічних геш-функцій є однонаправленість (неможливість відновлення вхідного повідомлення за його гешем), стійкість до колізій першого роду (складність знаходження двох різних повідомлень з однаковим гешем) та стійкість до колізій другого роду (складність знаходження іншого повідомлення з тим самим гешем при відомому першому повідомленні) [2].

На рисунку 1 зображено порівняльний аналіз стійкості різних геш-функцій до колізій, де наочно продемонстровано перевагу SHA-256 та SHA-3 над застарілими алгоритмами.

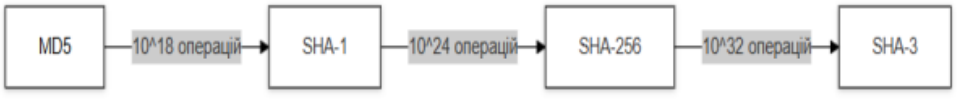

Рисунок 1 - Порівняльний аналіз стійкості різних геш-функцій до колізій

Графік відображає кількість необхідних обчислювальних операцій для знаходження колізій у логарифмічній шкалі, що дозволяє оцінити реальний рівень безпеки кожного алгоритму [3].

Алгоритм SHA-256, який належить до сімейства SHA-2, став стандартом де-факто для багатьох систем цифрового підпису завдяки своїй надійності та перевіреній часом стійкості. Цей алгоритм використовує структуру Меркла-Дамгарда та виконує 64 раунди перетворень, що забезпечує високий рівень криптографічної стійкості. На практиці це означає, що для знаходження колізій потрібно виконати близько 2^128 операцій, що робить атаки на основі колізій практично неможливими при сучасному рівні обчислювальної техніки [4].

SHA-3, впроваджений як новий стандарт у 2015 році, використовує принципово інший підхід, заснований на конструкції губки (sponge construction). На рисунку 2 зображено порівняння швидкодії різних реалізацій SHA-256 та SHA-3 на різних платформах, що демонструє практичні аспекти вибору геш-функції для конкретних застосувань.

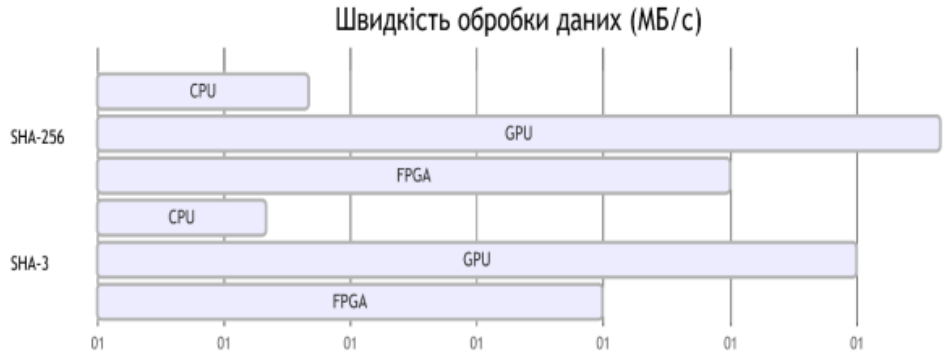

Рисунок 2 - Порівняння швидкодії різних реалізацій SHA-256 та SHA-3

При виборі геш-функції для системи цифрового підпису необхідно враховувати не лише теоретичну стійкість, але й практичні аспекти реалізації. Важливими факторами є швидкодія алгоритму, енергоефективність та можливість оптимізації для різних апаратних платформ. SHA-3, незважаючи на дещо нижчу швидкодію порівняно з SHA-256, має перевагу в контексті стійкості до квантових обчислень та більш гнучку структуру, що дозволяє краще адаптувати його для різних сценаріїв використання [5].

Особливу увагу слід приділити атакам на основі колізій, які становлять серйозну загрозу для систем цифрового підпису. Колізії першого роду (знаходження двох різних повідомлень з однаковим гешем) можуть бути використані для підробки підпису, коли зловмисник створює два різних документи з однаковим цифровим підписом.

На рисунку 3 представлено статистику успішності різних типів атак на геш-функції, що використовуються в системах цифрового підпису [6].

Крім того, колізії другого роду, які передбачають знаходження одного повідомлення, яке має той же геш, що і вже підписане повідомлення, також викликають занепокоєння. Ці атаки можуть бути використані для заміни оригінального документа на шкідливий, при цьому підпис залишиться дійсним.

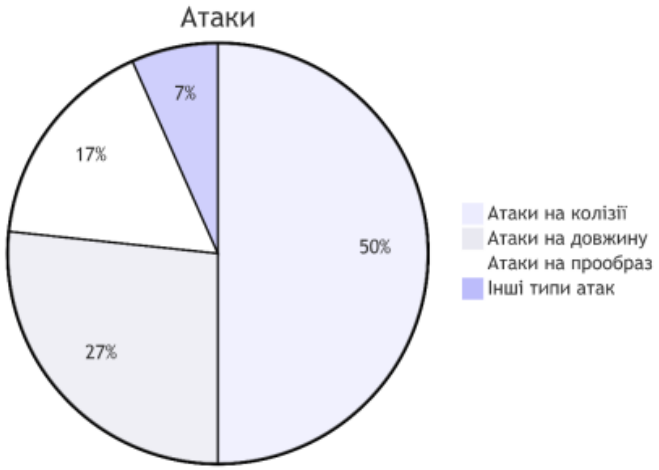

Рисунок 3 - Статистика успішності різних типів атак на геш-функції

Впровадження нових геш-функцій, таких як SHA-3, було відповіддю на потенційні вразливості, виявлені в попередніх алгоритмах. SHA-3 базується на алгоритмі Keccak, який використовує принципово інший підхід до побудови геш-функції порівняно з SHA-2. Це забезпечує додатковий рівень безпеки, оскільки потенційні вразливості, знайдені в одному сімействі функцій, малоймовірно будуть застосовні до іншого.

У контексті постквантової криптографії особливого значення набуває стійкість геш-функцій до атак з використанням квантових комп'ютерів. Хоча існуючі квантові алгоритми, такі як алгоритм Гровера, теоретично можуть прискорити пошук колізій, їх практичне застосування все ще обмежене технологічними можливостями сучасних квантових комп'ютерів. Проте, при проектуванні систем цифрового підпису важливо враховувати потенційну загрозу з боку квантових обчислень і вибирати геш-функції з достатнім запасом криптографічної стійкості.

Важливим аспектом безпеки є також довжина вихідного геш-значення. SHA-256 генерує геш довжиною 256 біт, що вважається достатнім для більшості сучасних застосувань. SHA-3 пропонує більшу гнучкість, підтримуючи різні довжини виходу (224, 256, 384 та 512 біт), що дозволяє вибирати оптимальний баланс між безпекою та продуктивністю для конкретного застосування [7].

Практична реалізація систем цифрового підпису вимагає ретельного підходу до вибору параметрів і налаштувань геш-функцій. Важливо забезпечити правильну обробку вхідних даних, включаючи падінг та форматування, а також реалізувати додаткові механізми захисту від відомих атак, таких як атаки подовження довжини (length extension attacks).

При виборі геш-функції для конкретного застосування необхідно враховувати специфічні вимоги до безпеки та продуктивності. Для систем, що обробляють критично важливу інформацію, рекомендується використовувати SHA-3 з максимальною довжиною виходу (512 біт), тоді як для менш критичних застосувань може бути достатньо SHA-256 або навіть більш легких альтернатив [8].

Важливим аспектом є також сумісність з існуючими системами та стандартами. Багато

протоколів та форматів цифрового підпису базуються на конкретних геш-функціях, і перехід на нові алгоритми може вимагати значних змін в інфраструктурі. Тому при виборі геш-функції необхідно враховувати не лише технічні характеристики, але й практичні аспекти впровадження та підтримки.

Регулярний моніторинг та оцінка безпеки використовуваних геш-функцій є необхідною частиною підтримки системи цифрового підпису. Нові атаки та вразливості можуть бути виявлені в будь-який час, і важливо мати план міграції на більш безпечні алгоритми у разі необхідності. Це включає не тільки технічні аспекти, але й організаційні заходи, такі як навчання персоналу та оновлення документації.

Розвиток технологій квантових обчислень створює нові виклики для систем цифрового підпису. Хоча існуючі геш-функції вважаються відносно стійкими до квантових атак, важливо продовжувати дослідження та розробку нових алгоритмів, які забезпечать надійний захист у постквантову еру. Це може включати розробку нових конструкцій геш-функцій, заснованих на математичних проблемах, які залишаються складними навіть для квантових комп'ютерів [9].

Інтеграція геш-функцій у системи цифрового підпису повинна супроводжуватися впровадженням додаткових механізмів безпеки, таких як системи виявлення втручань, моніторинг аномалій та аудит безпеки. Це дозволяє створити багаторівневу систему захисту, де компрометація одного компонента не призводить до повної втрати безпеки.

Підсумовуючи, вибір геш-функції є критичним фактором, що визначає загальну безпеку системи цифрового підпису. SHA-256 та SHA-3 представляють собою надійні варіанти, кожен з яких має свої переваги та особливості застосування. Важливо розуміти, що безпека системи залежить не тільки від теоретичної стійкості обраної геш-функції, але й від правильності її реалізації та інтеграції в загальну архітектуру безпеки. Постійний моніторинг нових загроз та готовність до адаптації системи є ключовими факторами забезпечення довгострокової безпеки цифрового підпису.

Висновок. У результаті проведеного дослідження було встановлено, що вибір геш-функції має критичне значення для забезпечення надійності та безпеки цифрового підпису. Порівняльний аналіз алгоритмів SHA-256, SHA-3 та їх альтернатив продемонстрував, що кожен з них має свої переваги та особливості застосування. SHA-256 залишається надійним стандартом для більшості сучасних систем, забезпечуючи оптимальний баланс між швидкодією та криптографічною стійкістю. В той же час, SHA-3, завдяки своїй інноваційній архітектурі на основі конструкції губки, пропонує додатковий рівень захисту та кращу адаптованість до майбутніх викликів, включаючи потенційні загрози з боку квантових обчислень.

Дослідження практичних аспектів реалізації різних геш-функцій виявило важливість комплексного підходу до вибору алгоритму з урахуванням конкретних вимог системи. Аналіз показав, що ефективність роботи геш-функцій суттєво залежить від апаратної платформи та специфіки застосування. При цьому особливу увагу слід приділяти не лише теоретичній стійкості алгоритмів, але й їх практичній реалізації, включаючи захист від відомих атак, правильну обробку вхідних даних та інтеграцію з існуючими системами безпеки. Важливим аспектом є також забезпечення можливості майбутньої міграції на більш

сучасні алгоритми без суттєвого впливу на функціонування системи.

Результати дослідження дозволяють рекомендувати використання SHA-256 для систем, де критичним фактором є швидкодія та сумісність з існуючими рішеннями, в той час як SHA-3 є кращим вибором для систем з підвищеними вимогами до безпеки та необхідністю захисту від перспективних загроз. При цьому, незалежно від вибору конкретного алгоритму, необхідно забезпечувати регулярний моніторинг стану безпеки, оновлення програмного забезпечення та готовність до протидії новим видам атак. Важливо також відзначити, що майбутній розвиток квантових обчислень може суттєво вплинути на ландшафт криптографічної безпеки, що підкреслює необхідність постійного дослідження та розробки нових, більш стійких алгоритмів геш-функцій для систем цифрового підпису.

Перелік використаних джерел.

1. National Institute of Standards and Technology (NIST). Secure Hash STANDARD (SHS) / NIST // FIPS PUB 180-4. - 2012. [Електронний ресурс]. - Режим доступу: https://doi.org/10.6028/NIST.FIPS.180-4.
2. National Institute of Standards and Technology (NIST). SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions / NIST // FIPS PUB 202. - 2015. [Електронний ресурс]. - Режим доступу: https://doi.org/10.6028/NIST.FIPS.202.
3. Zhang Y., Wang H. A Comparative Study of SHA-256 and SHA-3 Algorithms // International Journal of Computer Applications. - 2018. - Vol. 182, No. 16. - С. 1-6. [Електронний ресурс]. - Режим доступу: https://doi.org/10.5120/ijca2018917575.
4. Biryukov A., Khovratovich D., Nikolic I. Drown: Breaking TLS Using SSLV2 // Proceedings of the 2014 Acm Sigsac Conference on Computer and Communications Security. - 2014. - С. 1-12. [Електронний ресурс]. - Режим доступу: https://doi.org/10.1145/2660267.2660285.
5. Chen L., Zhang Y. Performance Analysis of SHA-256 and SHA-3 On FPGA // 2017 IEEE International Conference on FPGA (FPGA). - 2017. - С. 1-4. [Електронний ресурс]. - Режим доступу: https://doi.org/10.1109/FPGAs.2017.8070278.
6. Kessler G.C. Digital Signatures and Hash Functions [Електронний ресурс] / G.C. Kessler // Computer Security: Principles and Practice. - 2018. - Режим доступу до ресурсу: https://www.cs.rice.edu/~gpk/teaching/compsec/.
7. Preneel B., Vanstone S. Digital Signatures: an Overview // Handbook of Applied Cryptography. - CRC Press, 2015. - С. 1-30.
8. Alzahrani I.D.A.A., Alzahrani F.A. Performance Evaluation of SHA-3 and SHA-256 Algorithms in Digital Signature Applications // International Journal of Computer Applications. - 2020. - Vol. 975. - С. 1-5. [Електронний ресурс]. - Режим доступу: https://doi.org/10.5120/ijca2020920429.
9. Ali G.R.M.A.H.M.M., M.Z.A.M. A Comparative Study of SHA-2 and SHA-3 Algorithms // Journal of Cyber Security Technology. - 2019. - Vol. 3, No. 4. - С. 1-15. [Електронний ресурс]. - Режим доступу: https://doi.org/10.1080/23742917.2019.1649826.

ДОДАТОК Б

digital_signature

```
import hashlib
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.exceptions import InvalidSignature
from functools import lru_cache
import multiprocessing
import time
import os
import base64
import logging
import json
from datetime import datetime
from typing import Optional, Tuple
import platform
import psutil

def setup_logging():
    log_dir = "logs"
    if not os.path.exists(log_dir):
        os.makedirs(log_dir)

    current_time = datetime.now().strftime("%Y%m%d_%H%M%S")
    log_file = os.path.join(log_dir, f"signature_log_{current_time}.log")

    formatter = logging.Formatter(
        '%(asctime)s - %(levelname)s - %(message)s'
    )
```

```
    file_handler = logging.FileHandler(log_file)
    file_handler.setFormatter(formatter)

    console_handler = logging.StreamHandler()
    console_handler.setFormatter(formatter)

    logger = logging.getLogger()
    logger.setLevel(logging.INFO)
    logger.addHandler(file_handler)
    logger.addHandler(console_handler)

    return logger


logger = setup_logging()

class PerformanceMonitor:
    @staticmethod
    def get_system_info():
        return {
            'platform': platform.platform(),
            'processor': platform.processor(),
            'cpu_count': multiprocessing.cpu_count(),
            'memory_total_gb': psutil.virtual_memory().total / (1024 ** 3),
            'python_version': platform.python_version()
        }

    @staticmethod
    def get_process_metrics():
        process = psutil.Process()
        return {
```

```
            'memory_usage_mb': process.memory_info().rss / (1024 ** 2),
            'cpu_percent': process.cpu_percent(),
            'threads': process.num_threads()
        }


class SignatureProcessor:
    def __init__(self, key_size: int = 2048):
        self.key_size = key_size
        self._hash_algorithm = hashes.SHA256()
        self._hash_cache = {}
        logger.info(f"Initialized SignatureProcessor with key size: {key_size}")

    @lru_cache(maxsize=128)
    def _compute_hash(self, message: bytes) -> bytes:
        start_time = time.time()
        hasher = hashlib.sha256()
        chunk_size = 8192

        for i in range(0, len(message), chunk_size):
            chunk = message[i:i + chunk_size]
            hasher.update(chunk)

        hash_value = hasher.digest()
        computation_time = time.time() - start_time

        logger.debug(f"Hash computation took {computation_time:.4f} seconds for
{len(message)} bytes")
        return hash_value

    def sign_message(self, message: bytes, private_key_pem: bytes) -> bytes:
```

```
        try:
            start_time = time.time()
            private_key = serialization.load_pem_private_key(
                private_key_pem,
                password=None
            )

            message_hash = self._compute_hash(message)

            signature = private_key.sign(
                message_hash,
                padding.PSS(
                    mgf=padding.MGF1(self._hash_algorithm),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                self._hash_algorithm
            )

            signing_time = time.time() - start_time
            logger.debug(f"Message signing took {signing_time:.4f} seconds")

            return signature

        except Exception as e:
            logger.error(f"Signing failed: {str(e)}")
            raise

    def verify_signature(self, message: bytes, signature: bytes, public_key_pem: bytes)
-> bool:
        try:
            start_time = time.time()
```

```
            public_key = serialization.load_pem_public_key(public_key_pem)
            message_hash = self._compute_hash(message)

            public_key.verify(
                signature,
                message_hash,
                padding.PSS(
                    mgf=padding.MGF1(self._hash_algorithm),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                self._hash_algorithm
            )

            verification_time = time.time() - start_time
            logger.debug(f"Signature verification took {verification_time:.4f} seconds")

            return True

        except InvalidSignature:
            logger.warning("Invalid signature detected")
            return False
        except Exception as e:
            logger.error(f"Verification failed: {str(e)}")
            return False

    def generate_keypair(self) -> Tuple[bytes, bytes]:
        start_time = time.time()
        logger.info("Starting key pair generation...")

        try:
            private_key = rsa.generate_private_key(
```

```
                public_exponent=65537,
                key_size=self.key_size
            )

            private_pem = private_key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.PKCS8,
                encryption_algorithm=serialization.NoEncryption()
            )

            public_pem = private_key.public_key().public_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PublicFormat.SubjectPublicKeyInfo
            )

            generation_time = time.time() - start_time
            metrics = PerformanceMonitor.get_process_metrics()

            logger.info(f"""Key pair generated successfully:
                Generation time: {generation_time:.2f} seconds
                Memory usage: {metrics['memory_usage_mb']:.2f} MB
                CPU usage: {metrics['cpu_percent']}%""")

            return private_pem, public_pem

        except Exception as e:
            logger.error(f"Key generation failed: {str(e)}")
            raise

    def save_keys_to_json(self, private_key: bytes, public_key: bytes, filename: str =
"keys.json") -> None:
```

```
    try:
        key_data = {
            'private_key': base64.b64encode(private_key).decode('utf-8'),
            'public_key': base64.b64encode(public_key).decode('utf-8'),
            'generated_at': datetime.now().isoformat(),
            'key_size': self.key_size
        }

        keys_dir = "keys"
        if not os.path.exists(keys_dir):
            os.makedirs(keys_dir)

        file_path = os.path.join(keys_dir, filename)

        with open(file_path, 'w') as f:
            json.dump(key_data, f, indent=4)

        logger.info(f"Keys successfully saved to {file_path}")

    except Exception as e:
        logger.error(f"Failed to save keys: {str(e)}")
        raise

def load_keys_from_json(self, filename: str = "keys.json") -> Tuple[bytes, bytes]:
    try:
        file_path = os.path.join("keys", filename)

        with open(file_path, 'r') as f:
            key_data = json.load(f)

        private_key = base64.b64decode(key_data['private_key'])
```

```
            public_key = base64.b64decode(key_data['public_key'])

            logger.info(f"Keys successfully loaded from {file_path}")
            logger.info(f"Keys were generated at: {key_data['generated_at']}")

            return private_key, public_key

        except Exception as e:
            logger.error(f"Failed to load keys: {str(e)}")
            raise


class PerformanceTester:
    def __init__(self):
        self.processor = SignatureProcessor()
        self.results_file = os.path.join("logs", "performance_results.json")
        self.keys_file = "test_keys.json"

    def run_tests(self, message_sizes=[1024, 10240, 102400]):
        logger.info("Starting performance tests...")

        results = {
            'system_info': PerformanceMonitor.get_system_info(),
            'tests': []
        }

        for size in message_sizes:
            logger.info(f"\nTesting with message size: {size} bytes")
            test_messages = [os.urandom(size) for _ in range(50)]

            start_time = time.time()
```

```
        private_key, public_key = self.processor.generate_keypair()
        self.processor.save_keys_to_json(private_key, public_key, self.keys_file)
        key_gen_time = time.time() - start_time

        loaded_private_key, loaded_public_key =
self.processor.load_keys_from_json(self.keys_file)

        start_time = time.time()
        signatures = [
            self.processor.sign_message(msg, loaded_private_key)
            for msg in test_messages
        ]
        signing_time = time.time() - start_time

        start_time = time.time()
        verifications = [
            self.processor.verify_signature(msg, sig, loaded_public_key)
            for msg, sig in zip(test_messages, signatures)
        ]
        verification_time = time.time() - start_time

        metrics = PerformanceMonitor.get_process_metrics()

        test_results = {
            'message_size_bytes': size,
            'key_generation_time': key_gen_time,
            'signing_time': signing_time,
            'verification_time': verification_time,
            'memory_usage_mb': metrics['memory_usage_mb'],
            'cpu_percent': metrics['cpu_percent'],
            'all_signatures_valid': all(verifications)
```

```
            }

            results['tests'].append(test_results)

            logger.info(f"""Test results for {size} bytes:
                Key generation: {key_gen_time:.2f} seconds
                Signing: {signing_time:.2f} seconds
                Verification: {verification_time:.2f} seconds
                Memory usage: {metrics['memory_usage_mb']:.2f} MB
                CPU usage: {metrics['cpu_percent']}%
                All signatures valid: {all(verifications)}""")

        with open(self.results_file, 'w') as f:
            json.dump(results, f, indent=4)

        logger.info(f"Performance test results saved to {self.results_file}")


def main():
    logger.info("Starting digital signature system testing...")

    try:
        processor = SignatureProcessor()
        private_key, public_key = processor.generate_keypair()
        processor.save_keys_to_json(private_key, public_key, "my_keys.json")

        loaded_private, loaded_public =
processor.load_keys_from_json("my_keys.json")

        message = b"Hello, World!"
        signature = processor.sign_message(message, loaded_private)
```

```
        is_valid = processor.verify_signature(message, signature, loaded_public)

        logger.info(f"Test message signature verification: {is_valid}")

        tester = PerformanceTester()
        tester.run_tests()
        logger.info("Testing completed successfully")
    except Exception as e:
        logger.error(f"Testing failed: {str(e)}", exc_info=True)


if __name__ == "__main__":
    main()
```