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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(освітній рівень)

на тему: "Аналіз та оцінка ефективності сучасних асиметричних криптографічних алгоритмів"

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

Керівник Кульчицький Т.Р.
підпис (прізвище та ініціали)

Нормоконтроль Тимощук Д. І.
підпис (прізвище та ініціали)

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

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2024

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

Факультет компютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека
(шифр і назва спеціальності)

Студенту Вихованцю Дмитру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Аналіз та оцінка ефективності сучасних асиметричних криптографічних алгоритмів

Керівник роботи Кульчицький Тарас , к.ю.н., старший викладач,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

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

2. Термін подання студентом завершеної роботи 09.06.2024

3. Вихідні дані до роботи Характеристики асиметричних криптографічних алгоритмів

4. Зміст роботи (перелік питань, які потрібно розробити)
Знайти та описати сучасні асиметричні криптографічні алгоритми.
Проаналізувати властивості, характеристики та принцип роботи асиметричних алгоритмів.
Розробити програму, що реалізує досліджуванні асиметричні криптоалгоритми та виконує заміри їх ефективності.
Безпека життєдіяльності, основи охорони праці.
Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обовязкових креслень, слайдів)
Тема, мета, задачі. Мета та завдання роботи. Основні поняття. Аналіз алгоритму RSA. Аналіз алгоритму DSA. Аналіз алгоритму ECDSA. Вибір середовища виконання. Результати перевірки. Висновки.

6. Консультанти розділів роботи

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

7. Дата видачі завдання 29.01.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	29.01.2024	Виконано
2.	Підбір джерел для аналізу алгоритмів асиметричного шифрування	25.01 – 05.02	Виконано
3.	Опрацювання джерел в галузі дослідження	06.02 – 20.02	Виконано
4.	Проведення аналізу асиметричних криптосистем	22.02 – 12.03	Виконано
5.	Програмна реалізація алгоритмів асиметричного шифрування та цифрового підпису	15.03 - 25.03	Виконано
6.	Створення програмної реалізації модуля заміру використання ресурсів системи	25.02 – 10.04	Виконано
7.	Оформлення розділу «Асиметричні криптосистеми та їх роль у інформаційній безпеці»	10.02 – 05.03	Виконано
8.	Оформлення розділу «АНАЛІЗ СУЧАСНИХ АСИМЕТРИЧНИХ КРИПТОАЛГОРИТМІВ»	26.03 – 04.05	Виконано
9.	Оформлення розділу «ПРАКТИЧНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СИМЕТРИЧНИХ АЛГОРИТМІВ ШИФРУВАННЯ»	12.04 - 20.04	
10.	Виконання завдання до підрозділу «Безпека життєдіяльності, основи охорони праці»	25.04 – 10.05	Виконано
11.	Оформлення кваліфікаційної роботи	23.05 – 08.06	Виконано
12.	Нормоконтроль	10.06 – 12.06	Виконано
13.	Перевірка на плагіат	12.06 – 17.06	Виконано
14.	Попередній захист кваліфікаційної роботи	18.06 – 21.06	Виконано
15.	Захист кваліфікаційної роботи	25.06.2024	

Студент _____ Вихованець Д. В.
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

Аналіз та оцінка ефективності сучасних асиметричних криптографічних алгоритмів// Кваліфікаційна робота ОР «Бакалавр» // Вихованець Дмитро Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБ-41 // Тернопіль, 2024 // с. 73, рис. 8, табл. 1, лістинги 16 , бібліогр. 12.

Ключові слова: КРИПТОГРАФІЯ, ЦИФРОВИЙ ПІДПИС, ВРАЗЛИВОСТІ, АСИМЕТРИЧНЕ ШИФРУВАННЯ, АУТЕНТИФІКАЦІЯ, ЦІЛІСНІСТЬ ІНФОРМАЦІЇ, ЗАХИСТ ІНФОРМАЦІЇ.

Дослідження присвячено аналізу та оцінці ефективності сучасних асиметричних криптографічних алгоритмів. Вибір надійного криптографічного алгоритму є критично важливим для забезпечення безпеки інформаційних систем, оскільки він безпосередньо впливає на захист конфіденційних даних.

У роботі розглянуто три популярні алгоритми для цифрових підписів: RSA, DSA та ECDSA. Теоретичний аналіз включав дослідження архітектури алгоритмів та їх ключових параметрів. Проведені експериментальні дослідження дозволили оцінити продуктивність алгоритмів з точки зору часу виконання, використання пам'яті та навантаження на процесор.

Отримані результати та розроблений підхід можуть бути корисними для системних адміністраторів та розробників з метою підвищення рівня захисту даних в інформаційних системах.

Висновки та рекомендації, представлені у роботі, сприятимуть підвищенню рівня безпеки інформаційних систем завдяки вибору найбільш ефективного асиметричного криптографічного алгоритму відповідно до специфічних вимог та обмежень.

ANNOTATION

Analysis and Evaluation of the Effectiveness of Modern Asymmetric Cryptographic Algorithms // Bachelor's Qualification Work // Dmytro Volodymyrovych Vykhovanets // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, Group SB-41 // Ternopil, 2024 // p. 73, fig. 8, tab. 1, listings 16, bibl. 12.

Keywords: CRYPTOGRAPHY, DIGITAL SIGNATURE, VULNERABILITIES, ASYMMETRIC ENCRYPTION, AUTHENTICATION, INFORMATION INTEGRITY, INFORMATION PROTECTION.

The research is dedicated to the analysis and evaluation of the effectiveness of modern asymmetric cryptographic algorithms. The choice of a reliable cryptographic algorithm is critically important for ensuring the security of information systems, as it directly impacts the protection of confidential data.

The work examines three popular digital signature algorithms: RSA, DSA, and ECDSA. The theoretical analysis included studying the architecture of the algorithms and their key parameters. Experimental studies were conducted to evaluate the performance of the algorithms in terms of execution time, memory usage, and CPU load.

The obtained results and the developed approach can be useful for system administrators and developers to enhance the level of data protection in information systems.

The conclusions and recommendations presented in the work will contribute to improving the security of information systems by selecting the most effective asymmetric cryptographic algorithm according to specific requirements and constraints.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

RSA – Rivest, Shamir, Adelman.

NIST – National Institute of Standards and Technology

DSA – Digital Signature Algorithm.

ECDSA – Elliptic Curve Digital Signature Algorithm.

IP – Internet Protocol

PGP – Pretty Good Privacy

IPSec – IP Security

DoS - Denial of Services

DDoS – Distributed Denial of Services

ВСТУП

У сучасному світі, де інформаційні технології розвиваються стрімкими темпами, а інформаційні потоки глобалізуються, захист конфіденційних даних стає критично важливим для безпеки інформаційних систем. Криптографія, що вивчає методи шифрування та розшифрування інформації, є фундаментом для забезпечення цілісності, автентичності та конфіденційності даних. Одним із провідних підходів у цій сфері є використання асиметричних криптографічних алгоритмів, які застосовують різні ключі для шифрування та розшифрування: відкритий ключ для шифрування і закритий ключ для розшифрування.

Асиметричні криптографічні алгоритми, такі як RSA, ECC (еліптичні криві) та їхні модифікації, зарекомендували себе як ефективні та надійні інструменти у різноманітних сферах застосування, від захисту персональних даних і забезпечення безпеки фінансових транзакцій до електронного документообігу та захисту конфіденційної комунікації. Завдяки використанню різних ключів для шифрування та розшифрування, асиметрична криптографія усуває ризики, повязані з передачею секретних ключів, які характерні для симетричних методів.

Проте, зі збільшенням обсягу даних та зростанням обчислювальних потужностей, виникає постійна потреба у вдосконаленні існуючих алгоритмів та розробці нових методів шифрування, здатних протистояти сучасним кіберзагрозам. Зокрема, з появою квантових компютерів, існуючі асиметричні алгоритми можуть стати вразливими, що підкреслює необхідність розвитку квантово-стійкої криптографії.

Метою цього дослідження є оцінка ефективності сучасних асиметричних криптографічних алгоритмів та визначення їх сильних і слабких сторін. У роботі розглядаються теоретичні аспекти асиметричної криптографії, аналізуються існуючі алгоритми, а також проводиться порівняльний аналіз їх продуктивності. При дослідженні було виконано наступні завдання:

– Огляд теоретичних основ асиметричної криптографії:

a) Вивчення історії та розвитку асиметричної криптографії.

b) Аналіз основних принципів та концепцій асиметричних алгоритмів.

– Класифікація та характеристика сучасних асиметричних алгоритмів:

a) Опис найбільш відомих та поширених асиметричних алгоритмів, таких як RSA, ECC (еліптичні криві)

b) Визначення їх основних параметрів, характеристик та областей застосування.

– Аналіз продуктивності асиметричних криптографічних алгоритмів:

a) Вимірювання швидкості виконання криптографічних операцій (шифрування, дешифрування, підписання, верифікація).

b) Оцінка ресурсів, необхідних для виконання алгоритмів (час виконання, використання памяті, енергоспоживання).

– Висновки та рекомендації:

a) Формулювання загальних висновків на основі проведеного аналізу.

b) Надання рекомендацій щодо вибору та застосування асиметричних алгоритмів у різних умовах та для різних цілей.

Актуальність теми цього дослідження обумовлена зростаючими кіберзагрозами, необхідністю забезпечення надійного захисту інформації в умовах сучасного цифрового середовища та підготовкою до майбутніх викликів, зокрема тих, що повязані з розвитком квантових технологій.

Предметом дослідження є сучасні асиметричні алгоритми шифрування, що використовуються для захисту даних в інформаційних системах. Ці алгоритми є ключовим елементом систем безпеки, оскільки вони забезпечують захист даних від несанкціонованого доступу та зловмисних атак, включаючи підміну даних, несанкціоноване розшифрування та злам ключів.

Обєктом дослідження є асиметричні алгоритми шифрування, а також їх характеристики, властивості та ефективність у різних умовах застосування. Особлива увага приділяється аналізу таких алгоритмів, як RSA, ECC.

1 АСИМЕТРИЧНІ КРИПТОСИСТЕМИ ТА ЇХ РОЛЬ У ІНФОРМАЦІЙНІЙ БЕЗПЕЦІ

1.1 Поняття та задачі інформаційної безпеки

Перед розглядом інформаційної безпеки варто дати визначення поняттю. Інформація. Цей термін використовується щодня у багатьох сферах людської діяльності, в тому числі у законодавстві. Закон України «Про інформацію» дає таке визначення цьому поняттю: «інформація — будь-які відомості та/або дані, які можуть бути збережені на матеріальних носіях або відображені в електронному вигляді;»[1]. Проте ми вважаємо його неповним і дещо скудним для теми що ми розглядаємо, краще дамо більш актуальне. Інформація — це дані про обєкти, явища, події, процеси та факти, які сприймаються і використовуються людьми для прийняття рішень, виконання завдань та комунікації. Інформація може мати різну форму представлення: текстову, числову, графічну, звукову, відео, двовимірні або тривимірні моделі продукції, тощо.

У інформаційних системах інформація визначається як відомості, що підлягають зберіганню, передаванню та обробці. Проте спершу варто було б дати визначення інформаційній системі. Закон України «Про Захист інформації в інформаційно-телекомунікаційних системах» визначає це так: «інформаційно-комунікаційна система — сукупність інформаційних та електронних комунікаційних систем, які у процесі обробки інформації діють як єдине ціле;»[2]. Проте це твердження дуже скупе на деталі, тому дамо більш розгорнуте і актуальне для нашої теми: Інформаційні системи — це сукупність апаратних і програмних засобів, призначених для збору, зберігання, обробки, передачі та відображення інформації. Вони забезпечують виконання багатьох функцій, таких як управління даними, автоматизація процесів, аналітика та підтримка прийняття рішень.

Інформаційна безпека є сукупністю заходів та стратегій, спрямованих на захист конфіденційності, цілісності та доступності інформації в системах

обробки даних. Основні аспекти інформаційної безпеки включають захист від несанкціонованого доступу, збереження конфіденційності даних, забезпечення цілісності інформації в умовах зміни та передачі, а також забезпечення доступності інформаційних ресурсів для авторизованих користувачів.

- Цілісність — Ця властивість визначається як стан, при якому інформація зберігається, передається та оброблюється в незмінному вигляді, без будь-яких несанкціонованих змін або спотворень.
- Конфіденційність — Це доступність інформації лише для авторизованих осіб або субєктів. Інформація з конфіденційністю має бути захищена від несанкціонованого доступу.
- Доступність — Ця властивість визначає, наскільки легко користувачі можуть отримати доступ до інформаційних ресурсів в потрібний момент часу. Забезпечення доступності гарантує, що інформаційні системи залишаються доступними для користувачів навіть у випадку збоїв чи атак.

Далі ми детальніше розглянемо кожну з цих властивостей інформації та способи їх забезпечення:

Інформаційна безпека в сучасних інформаційних системах спрямована на забезпечення захисту від різноманітних загроз, які можуть впливати на їхню працездатність та безпеку даних. Однією з основних загроз є кібератаки, до яких належать віруси, черви та троянські коні. Віруси здатні самовідтворюватися і заражати файли та системи, поширюючись на інші компютери. Черви, на відміну від вірусів, розповсюджуються через мережі без втручання користувача, тоді як троянські коні маскуються під легітимне програмне забезпечення, виконуючи небажані дії на компютері жертви, такі як крадіжка даних або створення бекдорів для подальших атак.

Витоки даних становлять іншу серйозну загрозу, яка включає несанкціонований доступ до конфіденційної інформації. Це може бути наслідком зламу паролів, використання вразливостей у програмному забезпеченні або застосування соціальної інженерії. Часто використовуваним методом отримання конфіденційної інформації є фішинг, де зловмисники видають себе за легітимні організації через електронну пошту або вебсайти.

Фізичні загрози також є важливим аспектом інформаційної безпеки. Пожежі, повені та крадіжки обладнання можуть призвести до фізичного знищення або недоступності інформаційної системи. Зокрема, пожежі та повені можуть завдати серйозних пошкоджень апаратному забезпеченню, тоді як крадіжки можуть призвести до втрати конфіденційної інформації та порушення функціонування системи.

Внутрішні загрози включають недбалість або зловмисні дії працівників. Недбалість може проявлятися у вигляді порушення політик безпеки, що призводить до випадкових витоків даних або порушення цілісності інформаційних систем. Зловмисні дії працівників можуть включати саботаж, шпигунство або крадіжку даних.

Атаки на доступність спрямовані на перевантаження ресурсів системи, що призводить до її недоступності для легітимних користувачів. DoS-атаки використовують надмірне навантаження на ресурси, тоді як DDoS-атаки залучають велику кількість скомпрометованих пристроїв для одночасного надсилання запитів на цільовий ресурс.

Соціально-інженерні атаки використовують психологічні методи для обману користувачів з метою отримання конфіденційної інформації або доступу до системи. Приклади таких атак включають фішинг, вішинг (фішинг по телефону) та смишинг (фішинг через текстові повідомлення). Інсайдерські загрози, які виникають внаслідок дій працівників, також можуть використовувати маніпуляції або недбалість для доступу до захищених систем.

Технічні вади програмного забезпечення, такі як уразливості в коді, некоректні налаштування систем або несвоєчасне оновлення програмного забезпечення, також становлять значну загрозу. Уразливості можуть бути використані зловмисниками для отримання несанкціонованого доступу або запуску шкідливих програм, тоді як некоректні налаштування та застаріле програмне забезпечення створюють лазівки для атак.

Забезпечення захисту від цих загроз та атак вимагає комплексного підходу, який включає технічні, організаційні та адміністративні заходи, а також постійне вдосконалення і оновлення систем безпеки.

Інформаційна безпека досить широке поняття, яке включає в себе безпеку не лише самої інформації чи ІКС, а й усієї інфраструктури загалом, в тому числі і мережі, фізичного доступу до носіїв чи систем інформації, а також захист структури від несанкціонованого доступу.

Інформаційна безпека є комплексною і багатогранною сферою діяльності, що вимагає всебічного та систематичного підходу для досягнення успіху. Для забезпечення безпеки в інформаційних системах необхідно виконувати низку завдань. Перш за все, захист інформації під час зберігання, обробки та передачі мережею є критично важливим. Це вимагає впровадження шифрування даних, використання захищених каналів звязку та резервного копіювання для запобігання втратам.

Також необхідно підтверджувати достовірність даних та автентифікацію користувачів. Використання механізмів автентифікації, таких як паролі, двофакторна автентифікація та біометричні дані, дозволяє підтвердити особистість користувачів та обєктів даних, що встановлюють звязок. Важливою задачею є виявлення та попередження порушень цілісності даних. Це можна досягти завдяки застосуванню алгоритмів контрольних сум, хешування та технологій моніторингу змін, що забезпечують цілісність даних.

Не менш важливим є захист технічних пристроїв та приміщень, що включає впровадження фізичної безпеки, контроль доступу до приміщень, використання замків, відеоспостереження та захисних екранів для технічних пристроїв. Захист конфіденційної інформації від витоку та вбудованих електронних пристроїв передбачає використання методів електромагнітної сумісності (EMC), екранування, а також програмного забезпечення для виявлення та блокування несанкціонованих пристроїв збору інформації.

Захист програмного забезпечення від шкідливих програм та вірусів вимагає впровадження антивірусного програмного забезпечення, регулярних оновлень систем, контролю доступу до програм та використання віртуалізації. Щодо захисту від несанкціонованого доступу до інформаційних ресурсів та мережевих засобів, то тут важливим є впровадження систем виявлення та

запобігання вторгнень (IDS/IPS), міжмережевих екранів (фаєрволів), контроль доступу до мережевих пристроїв та шифрування трафіку.

Організація заходів для забезпечення збереження конфіденційних даних включає розробку та впровадження політик безпеки, навчання персоналу, проведення регулярних аудитів та оцінок ризиків, а також впровадження систем управління інформаційною безпекою (ISMS).

Насамкінець, варто підкреслити, що інформаційна безпека є критично важливою складовою сучасного цифрового суспільства. Вона забезпечує захист конфіденційності, цілісності та доступності інформації. В умовах постійного розвитку технологій та зростання кількості кіберзагроз ефективне управління інформаційною безпекою стає життєво необхідним для організацій та окремих осіб, сприяючи захисту їх даних та підтримці безперервної діяльності.

1.2 Роль асиметричних криптосистем в інформаційній безпеці

Асиметричні криптосистеми відіграють ключову роль в інформаційній безпеці завдяки своїм унікальним властивостям, що дозволяють забезпечити конфіденційність, цілісність та автентичність даних. Основні аспекти їхньої ролі включають шифрування та дешифрування даних, створення електронних підписів, управління ключами, ідентифікацію та аутентифікацію, а також використання цифрових сертифікатів. Розглянемо кожен з цих аспектів детальніше.

Почнемо з шифрування та дешифрування. Асиметричні криптосистеми використовують пару ключів: відкритий та закритий. Дані, зашифровані відкритим ключем, можуть бути дешифровані лише за допомогою відповідного закритого ключа. Це забезпечує конфіденційність інформації, оскільки навіть якщо зашифровані дані будуть перехоплені, без закритого ключа їх неможливо прочитати. Відкритий ключ можна вільно передавати через незахищені канали звязку, тоді як закритий ключ зберігається в таємниці. Такий підхід вирішує проблему безпечної передачі ключів, характерну для симетричних

криптосистем, де один ключ використовується для шифрування та дешифрування. Приклад різниці цих підходів наведено на рисунку 1.1.

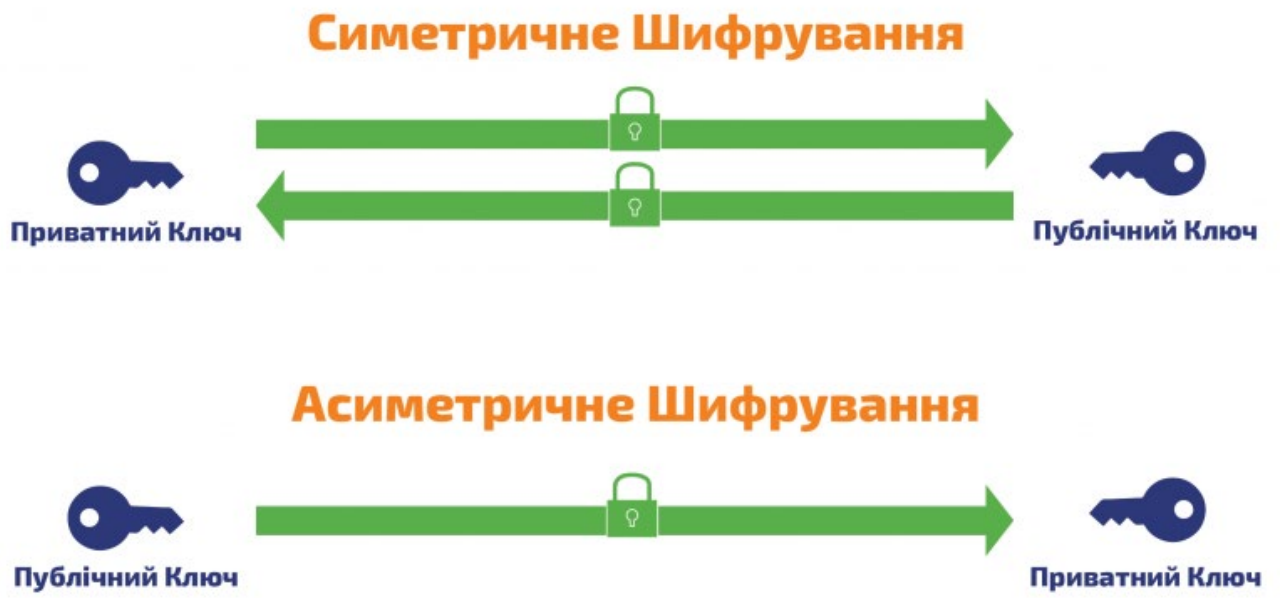

Рисунок 1.1 – Відмінність симетричного та асиметричного шифрування

Наступний важливий аспект – електронний підпис. Електронний підпис створюється шляхом шифрування гешу (хешу) документа за допомогою закритого ключа. Отримувач може перевірити цей підпис, розшифрувавши геш за допомогою відкритого ключа відправника та порівнявши його з гешем отриманого документа. Це гарантує цілісність даних, підтверджуючи, що документ не був змінений після підпису, та автентичність, підтверджуючи особу відправника. Така перевірка електронного підпису широко використовується в юридичних та фінансових транзакціях, де необхідно забезпечити високий рівень безпеки.

Управління ключами в асиметричних криптосистемах значно спрощує процес розподілу криптографічних ключів. Відкриті ключі можуть бути передані через незахищені канали, тоді як закриті ключі залишаються в секреті у своїх власників. Це вирішує проблему захищеної передачі ключів, яка є важливою в симетричних криптосистемах. Асиметричні криптосистеми використовуються для створення та обміну сеансових ключів у протоколах захищеної передачі

даних, таких як SSL/TLS, де асиметричне шифрування використовується на початковому етапі для встановлення захищеного зєднання, а потім для шифрування даних використовується швидке симетричне шифрування.

Ідентифікація та аутентифікація також є важливою частиною використання асиметричних криптосистем. Використання пари ключів дозволяє здійснювати двосторонню аутентифікацію між користувачами та серверами, забезпечуючи захищені зєднання та підтвердження особи обох сторін. Наприклад, у протоколах SSL/TLS, які використовуються для захисту веб-трафіку, сертифікати SSL, підписані на основі асиметричних ключів, підтверджують автентичність веб-сайту користувачеві, забезпечуючи захист від атак "людина посередині" (Man-in-the-Middle).

Асиметричне шифрування відіграє критичну роль у блокчейн-технологіях, таких як Bitcoin та інші криптовалюти. В основі цих систем лежить концепція децентралізованого реєстру, де всі транзакції записуються у вигляді послідовності блоків, обєднаних у єдиний ланцюг. Використання асиметричного шифрування забезпечує кілька важливих аспектів інформаційної безпеки: автентифікацію користувачів, цілісність даних та конфіденційність транзакцій.

У блокчейн-системах кожен користувач має пару криптографічних ключів: відкритий і закритий ключі. Закритий ключ зберігається в таємниці і використовується для підпису транзакцій, тоді як відкритий ключ публічно доступний і використовується іншими користувачами для перевірки підпису. Коли користувач хоче здійснити транзакцію, він підписує її своїм закритим ключем. Підпис включається у транзакцію разом з відкритим ключем користувача, що дозволяє будь-якому учаснику мережі перевірити дійсність підпису і, відповідно, автентичність транзакції. Це забезпечує захист від підробки транзакцій і гарантує, що тільки власник відповідного закритого ключа міг ініціювати транзакцію.

Цілісність даних у блокчейн-технологіях забезпечується за допомогою хешування. Кожен блок у ланцюзі містить хеш попереднього блоку, а також хеш своїх власних даних, включаючи транзакції, що забезпечує звязок між блоками. Будь-яка зміна у даних будь-якого блоку змінить його хеш, що призведе до

невідповідності з хешем, збереженим у наступному блоці. Це забезпечує незмінність блокчейну: якщо дані одного з блоків будуть змінені, це одразу стане очевидним для всіх учасників мережі. Тому цілісність і незмінність ланцюга блоків гарантуються криптографічно.

Крім того, асиметричне шифрування забезпечує конфіденційність у блокчейн-системах. Хоча самі дані транзакцій у публічних блокчейнах можуть бути доступними для всіх, використання криптографічних ключів забезпечує анонімність користувачів. Відкриті ключі, які виступають як адреси користувачів, не розкривають особистих даних, і тільки власники відповідних закритих ключів можуть ініціювати транзакції з цих адрес. Це дозволяє зберігати конфіденційність фінансової інформації, одночасно забезпечуючи прозорість транзакцій.

Прикладом використання асиметричного шифрування у блокчейн-технологіях є Bitcoin. Кожна транзакція в Bitcoin підписується закритим ключем відправника, що забезпечує автентифікацію і підтвердження права власності на кошти. Хеш кожного блоку включає хеш попереднього блоку, створюючи ланцюг, який забезпечує цілісність усієї історії транзакцій. Інший приклад – Ethereum, платформа, що дозволяє створювати смарт-контракти. У Ethereum також використовуються криптографічні ключі для підпису транзакцій і забезпечення автентичності, а хешування забезпечує цілісність блокчейну.

Цифрові сертифікати є основою для інфраструктури відкритих ключів (PKI). PKI включає сертифікаційні центри (CA), які видають цифрові сертифікати, що підтверджують автентичність відкритих ключів та ідентифікацію їх власників. Цифрові сертифікати використовуються в багатьох захищених протоколах, таких як SSL/TLS для веб-безпеки, S/MIME для захищеної електронної пошти, та в багатьох інших системах, де необхідна безпека комунікацій та даних. PKI також використовується в протоколах IPsec для створення віртуальних приватних мереж (VPN), забезпечуючи захищений обмін даними між різними мережами через незахищені канали звязку.

PGP (Pretty Good Privacy), який використовується для захисту електронної пошти. PGP забезпечує конфіденційність і автентичність повідомлень,

використовуючи комбінацію асиметричного та симетричного шифрування. Відправник шифрує повідомлення симетричним ключем, який потім шифрується відкритим ключем отримувача. Отримувач розшифровує симетричний ключ своїм закритим ключем і використовує його для дешифрування повідомлення. PGP також підтримує цифрові підписи для перевірки автентичності повідомлень.

IPsec (Internet Protocol Security) – ще один важливий протокол, який використовує асиметричне шифрування. IPsec забезпечує захищений обмін даними на рівні мережевого протоколу IP, часто використовується для створення віртуальних приватних мереж (VPN). IPsec включає два основних протоколи: AH (Authentication Header) та ESP (Encapsulating Security Payload). AH забезпечує автентифікацію і цілісність даних, а ESP забезпечує конфіденційність, автентифікацію та цілісність. Асиметричне шифрування в IPsec використовується під час встановлення сеансових ключів для забезпечення безпеки каналів звязку.

SSH (Secure Shell) – протокол, що забезпечує захищений доступ до віддалених компютерів. SSH використовує асиметричне шифрування для автентифікації користувачів та встановлення захищеного каналу звязку. Під час ініціації зєднання клієнт і сервер обмінюються відкритими ключами для створення спільного секрету, який використовується для симетричного шифрування подальших даних.

S/MIME (Secure/Multipurpose Internet Mail Extensions) – стандарт для шифрування та підпису електронної пошти. S/MIME використовує сертифікати, видані сертифікаційними центрами, для забезпечення автентичності та конфіденційності повідомлень. Відправник шифрує повідомлення за допомогою симетричного ключа, який потім шифрується відкритим ключем отримувача. Отримувач використовує свій закритий ключ для дешифрування симетричного ключа і, відповідно, повідомлення.

Протокол Kerberos також використовує асиметричне шифрування для автентифікації користувачів і серверів у мережі. Kerberos заснований на системі квитків (tickets), які видаються централізованим сервером автентифікації (KDC

– Key Distribution Center). Квитки шифруються ключами, що забезпечує їхню безпеку і цілісність.

DNSSEC забезпечує автентичність і цілісність даних DNS. Кожна DNS-зона підписується приватним ключем, а публічний ключ розповсюджується через DNS. Коли користувач запитує DNS-запис, він також отримує відповідний цифровий підпис, який може перевірити за допомогою публічного ключа. Це гарантує, що відповідь не була змінена.

DANE використовує DNSSEC для поширення сертифікатів X.509 або публічних ключів, які використовуються в протоколах TLS. Це дозволяє уникнути залежності від традиційних сертифікаційних центрів, оскільки публічні ключі або сертифікати можуть бути поширені через захищені DNS-записи.

SPF дозволяє власникам доменів визначати, які сервери мають право надсилати електронну пошту від їхнього імені, що допомагає запобігати фішинг-атакам. DKIM дозволяє доменам підписувати вихідні повідомлення за допомогою приватного ключа, забезпечуючи автентичність і цілісність. Одержувач може перевірити підпис за допомогою публічного ключа, опублікованого в DNS.

EAP-TLS – це метод автентифікації, який використовується в бездротових мережах і VPN. Він забезпечує захищене зєднання за допомогою TLS, використовуючи сертифікати для автентифікації клієнта і сервера. Під час встановлення зєднання клієнт і сервер обмінюються сертифікатами і використовують асиметричне шифрування для створення сеансових ключів.

WPA2-Enterprise використовує EAP-TLS для автентифікації користувачів у бездротових мережах. Кожен користувач має сертифікат, який використовується для автентифікації та встановлення захищеного зєднання з мережею. Це забезпечує високий рівень безпеки порівняно зі спільними паролями.

Signal Protocol використовується для забезпечення конфіденційності повідомлень у сучасних месенджерах, таких як Signal та WhatsApp. Протокол використовує комбінацію асиметричного та симетричного шифрування для

забезпечення end-to-end шифрування. Кожне повідомлення шифрується окремо, що забезпечує високий рівень безпеки навіть у випадку компрометації ключів.

ISO/IEC 9798 – це міжнародний стандарт, що визначає протоколи автентифікації на основі асиметричних криптосистем. Протоколи цього стандарту використовуються для перевірки особи в різних системах безпеки, забезпечуючи захист від атак на автентифікацію та цілісність.

Усі ці протоколи демонструють важливість асиметричних криптосистем у забезпеченні без

Таким чином, асиметричні криптосистеми є невідємною частиною сучасних систем інформаційної безпеки, забезпечуючи надійний захист даних, аутентифікацію користувачів та цілісність інформації у різних протоколах та застосуваннях.

2 АНАЛІЗ СУЧАСНИХ АСИМЕТРИЧНИХ КРИПТОАЛГОРИТМІВ

У цьому розділі буде розглянуто теоретичні аспекти використання асиметричних криптоалгоритмів у захисті інформації. Хоча такі алгоритми є досить ефективними та безпечними для шифрування даних і стійкими до різних типів атак ми сконцентруємо свою увагу на іншому способі їх використання, а саме на цифровому підписі для забезпечення аутентифікації та цілісності даних при передачі.

Для вибору великого простого числа в асиметричних криптосистемах існують кілька підходів, кожен з яких має свої переваги та недоліки. Одним з основних методів є генерація випадкових чисел з наступною перевіркою їх на простоту. Цей метод дозволяє отримати числа необхідної довжини, що є важливою вимогою для забезпечення високого рівня безпеки.

Процес починається з генерації випадкового числа необхідної довжини, наприклад, 2048 біт. Після цього число перевіряється на простоту. Одним з найпоширеніших способів перевірки є використання тесту Міллера-Рабіна. Це ймовірнісний тест, який перевіряє, чи є число складеним, використовуючи розкладання n-1 на множники вигляду 2^s×d, де s —це додатне ціле число, а d – це непарне число. Тест Міллера-Рабіна має високу точність і може бути налаштований на різний рівень точності шляхом збільшення кількості ітерацій.

Іншим методом є тест Ферма, який базується на малому теоремі Ферма. За цією теоремою, якщо p — просте число, то для будь-якого цілого числа a, яке не є дільником p, виконується рівність a^(p-1)=1 mod p. Однак тест Ферма менш надійний через існування чисел Кармайкла, які можуть пройти тест, не будучи простими.

Ще одним ймовірнісним тестом є тест Соловея-Штрассена, який використовує символ Якобі для перевірки простоти числа. Цей тест порівнює значення символу Якобі і результат піднесення до степеня, що дозволяє більш точно визначити, чи є число простим.

Для гарантованого визначення простоти числа можна використовувати детермінований алгоритм АКС (Агравал-Кайал-Саксена). Цей алгоритм має

поліноміальну часову складність і гарантує точність, але є складнішим у реалізації і має вищу обчислювальну вартість.

Вибір методу залежить від вимог до безпеки та ресурсів. У більшості випадків для вибору великого простого числа p в алгоритмі Діффі-Геллмана використовують комбінацію генерації випадкових чисел і ймовірнісних тестів, таких як тест Міллера-Рабіна, для забезпечення високої точності та ефективності.

2.1 Алгоритм RSA

RSA (Rivest–Shamir–Adleman) – це криптосистема з відкритим ключем, одна з найстаріших і найширше використовуваних для забезпечення безпеки передачі даних. Назва "RSA" походить від прізвищ Рона Рівеста, Аді Шаміра та Леонарда Адлемана, які вперше описали цей алгоритм у 1977 році. Аналогічна система була розроблена секретно в 1973 році в урядовій організації з комунікацій Великої Британії (GCHQ) математиком Кліффордом Коксом, і ця розробка була розсекречена у 1997 році.

У криптосистемі з відкритим ключем ключ шифрування є загальнодоступним і відрізняється від ключа розшифрування, який зберігається в секреті (приватний). Користувач RSA створює та публікує відкритий ключ, який базується на двох великих простих числах, а також додаткове значення. Прості числа зберігаються в секреті. Повідомлення можуть бути зашифровані будь-ким, використовуючи відкритий ключ, але можуть бути розшифровані лише тим, хто знає приватний ключ.

Безпека RSA залежить від практичної складності факторизації добутку двох великих простих чисел – "проблеми факторизації". Зламати шифрування RSA відомо як проблема RSA. Відкрите питання, чи є ця проблема такою ж складною, як і проблема факторизації. На даний момент немає опублікованих методів зламу системи при використанні досить великого ключа.

RSA – відносно повільний алгоритм. Через це він рідко використовується для безпосереднього шифрування даних користувачів. Найчастіше RSA

використовується для передачі спільних ключів для симетричної криптографії, які потім використовуються для масового шифрування-розшифрування.

Ідея асиметричної криптосистеми з відкритим і приватним ключами належить Вітфілду Діффі та Мартіну Геллману. Вони також ввели поняття цифрових підписів і намагалися застосувати теорію чисел. Їх формулювання використовувало спільний секретний ключ, створений за допомогою піднесення до степеня деякого числа за модулем простого числа. Проте вони не вирішили проблему реалізації односторонньої функції, можливо, через недостатньо вивчену на той час складність факторизації.

Основний принцип RSA алгоритму полягає в спостереженні, що практично можливо знайти три дуже великі позитивні цілі числа , d та n, такі що для всіх цілих чисел m ($0 \leq m < n$), як m мають той самий залишок при діленні на n (тобто вони конгруентні за модулем n), приклад наведено у формулі (2.1)

$$(m^e)^d = m(mod\, n) \tag{2.1}$$

Відомо, що при наявності лише значень e та n надзвичайно складно знайти d. Цілі числа n та e формують публічний ключ, d є приватним ключем, а m — це повідомлення. Модульна експонентація до e і d відповідає шифруванню та дешифруванню відповідно.

Крім того, оскільки експоненти можуть бути змінені місцями, приватний і публічний ключі можуть також бути змінені місцями, дозволяючи підписувати та перевіряти повідомлення, використовуючи той самий алгоритм.

Далі буде наведено алгоритм генерації ключів у RSA:

– Виберемо два великі прості числа p і q. Щоб ускладнити факторизацію, p і q мають бути обрані випадковим чином, бути великими і мати значну різницю. Для їх вибору використовують стандартний метод вибору випадкових чисел та тест на простоту до знаходження двох простих чисел. p і q повинні зберігатися в таємниці.

– Обчислимо n за формулою 2.2. n використовується як модуль для обох ключів. Його довжина, зазвичай виражена в бітах, є довжиною ключа. n публікується як частина публічного ключа.

$$n = pq \tag{2.2}$$

– Обчислимо $\varphi(n)$ (функцію Ейлера) за формулою 2.3. Цей параметр зберігається в таємниці.

$$\varphi(n) = (\rho - 1)(q - 1) \tag{2.3}$$

– Виберемо ціле число e таке, що$1 < e < \varphi(n)$ і $gcd\big(e, \varphi(n)\big) = 1$, тобто e та $\varphi(n)$ є взаємно простими. Вибір e з короткою довжиною біта і малою кількістю одиниць у двійковому поданні робить шифрування більш ефективним – найчастіше використовуваним значенням для e є 2^{16}+1=65537. Найменше можливе значення для e – 3, але воно виявилося менш безпечним у деяких випадках. e публікується як частина публічного ключа.

– Визначимо d за формуло 2.4 тобто, d є модульним мультиплікативним оберненим до e за модулем $\varphi(n)$. Інакше кажучи, розязавши рівняння 2.5 отримаємо саме d. d можна ефективно обчислити, використовуючи розширений алгоритм Евкліда. d зберігається в таємниці як приватний ключ.

$$d = e^{-1}(mod\, \varphi(n)) \tag{2.4}$$

$$de = 1(mod\, \varphi(n)) \tag{2.5}$$

Отож використавши цей алгоритм отримаємо наступні дані — Публічний ключ, що складається з модуля n та публічної експоненти e, приватний ключ, що складається з приватної експоненти d, яка повинна зберігатися в таємниці. p, q і

$\varphi(n)$ також повинні зберігатися в таємниці, оскільки вони можуть бути використані для обчислення d.

Припустимо, що Аліса хоче відправити інформацію Бобу. Якщо вони вирішують використовувати RSA, Аліса повинна підписати дані своїм приватним ключем, а Боб повинен використовувати її публічний ключ для перевірки підпису повідомлення. Щоб Аліса могла відправляти свої підписані повідомлення, Боб отримує її публічний ключ через надійний, але не обовязково секретний канал. Приватний ключ Аліси (d) ніколи не розповсюджується.

Отримавши публічний ключ Аліси, Боб може відправити повідомлення m до Аліси. Для цього він спершу обчисл.є хеш повідомлення $H(m)$. Підпис створюється шляхом піднесення хеш-значення до степеня d за модулем n формула (2.6).

$$S = H(m) \bmod h \tag{2.6}$$

Отримуємо повідомлення m та підпис S. Тепер Аліса може відправити повідомлення m та підпис S

Аліса може перевірити підпис повідомлення S за формуло. (2.7).

$$H(m) = S^e \bmod n \tag{2.7}$$

Для наглядності оберемо прості параметри криптосистеми та пройдемося покроково по алгоритму, викону.чи усі операції у наведеній послідовності:

- Оберемо два різних простих числа, наприклад $p = 61$ та $q = 53$
- Обчислимо n за формулою 2.2, що дає нам $n = 61 \times 53 = 3233$
- Обчислимо функцію Ейлера для n за формулою 2.3 і отримаємо $\varphi(n) = (61 - 1) - (53 - 1) = 3120$.
- Виберемо випадкове просте число $e = 17$
- Обчислимо d модульний мультиплікативний обернений до e за формулою 2.4 та отримаємо $d = 2753$
- Публічний ключ складається з $(n = 3233, e = 17)$.

– Приватний ключ буде складатися з ($n = 3233, d = 2753$).

Далі слідує підпис повідомлення та перевірка підпису, проте вони будуть більш детально розглянуті у подальших розділах цієї роботи.

Схематичне зображення алгоритму цифрового підпису RSA наведено на рисунку 2.1

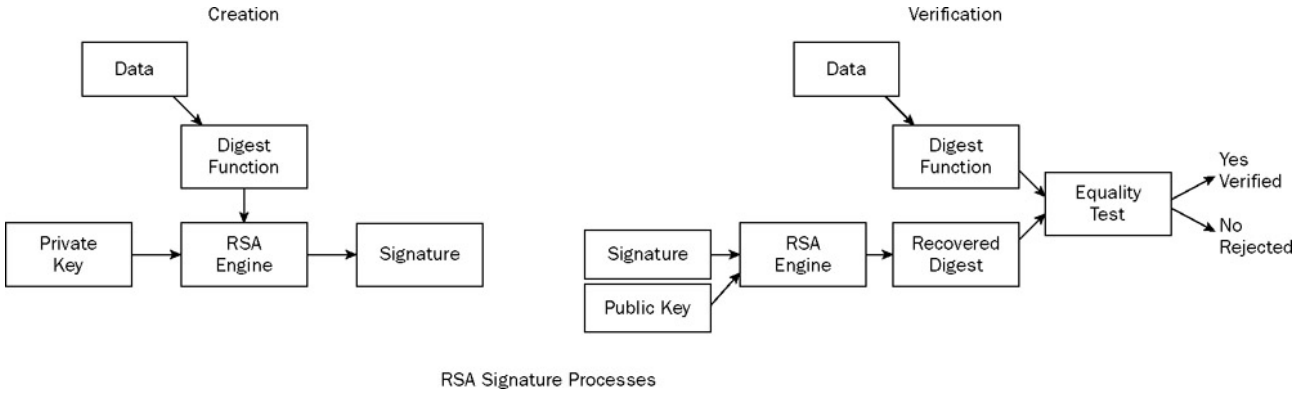

Рисунок 2.1 – Схема алгоритму цифрового підпису RSA

Оцінюючи цей алгоритм до переваг можна віднести:

– RSA забезпечує високий рівень безпеки при використанні великих ключів (2048 біт або більше). Підписання і перевірка підписів за RSA покладаються на складність факторизації великих чисел.Відкритий стандарт: RSA є відкритим і широко визнаним стандартом, що сприяє його інтеграції в різні криптографічні протоколи та системи.
– RSA є відкритим і широко визнаним стандартом, що сприяє його інтеграції в різні криптографічні протоколи та системи. Це полегшує його використання в різних додатках і забезпечує сумісність між різними системами.
– RSA для цифрового підпису використовується в багатьох криптографічних протоколах, таких як SSL/TLS, PGP, SSH, що робить його одним з найпоширеніших методів цифрового підпису.
– Цифровий підпис RSA забезпечує аутентифікацію підписувача та гарантує цілісність підписаного повідомлення. Це означає, що одержувач може бути впевнений, що повідомлення не було змінене і що воно було підписане саме тим, хто має приватний ключ.

В свою чергу до недоліків віднесемо наступне:

- Процес створення і перевірки цифрового підпису RSA є обчислювально складним, особливо для великих ключів. Це може призводити до повільної роботи в системах з обмеженими ресурсами, таких як мобільні пристрої або вбудовані системи.
- Для забезпечення надійної безпеки потрібні великі ключі (2048 біт або більше), що збільшує вимоги до зберігання і передачі. Це може бути проблематично в середовищах з обмеженими ресурсами.
- Квантові компютери можуть потенційно зламати RSA за допомогою алгоритму Шора, що ставить під загрозу довгострокову безпеку. В майбутньому можуть знадобитися альтернативні алгоритми, стійкі до квантових атак.
- Неправильне впровадження або налаштування RSA для цифрового підпису може призвести до вразливостей, які знизять безпеку. Важливо дотримуватись найкращих практик і стандартів при впровадженні RSA.

RSA є потужним і широко використовуваним алгоритмом асиметричної криптографії. Його безпека ґрунтується на складності факторизації великих чисел, що забезпечує надійний захист інформації при використанні великих ключів. Втім, обчислювальна складність і вимоги до великих ключів є його основними недоліками. Відомі атаки на алгоритм, як правило, спрямовані на слабкі ключі або специфічні вразливості, тому правильна реалізація та використання великих ключів є ключовими для забезпечення безпеки.

2.2 Алгоритм цифрового підпису DSA

Алгоритм цифрового підпису DSA (Digital Signature Algorithm) був розроблений в 1991 році Національним інститутом стандартів і технологій США (NIST). Його основною метою було забезпечення безпечної та ефективної цифрової підписки документів, яка могла б використовуватися у різних сферах, таких як електронна комерція, урядові документи та інші цифрові транзакції. DSA базується на алгоритмі дискретного логарифмування, подібному до

алгоритму Діффі-Хеллмана, який використовується для захисту переданих даних.

Основна ідея DSA полягає в створенні цифрового підпису, який базується на криптографічних принципах дискретного логарифмування. Особливістю цього алгоритму є використання двох ключів: приватного для підписання повідомлень та публічного для перевірки підпису. Це дозволяє забезпечити автентичність і цілісність повідомлень, оскільки тільки власник приватного ключа може створити дійсний підпис, який може бути перевірений за допомогою відповідного публічного ключа.

Відмінність DSA від інших алгоритмів цифрового підпису, таких як RSA, полягає в різних математичних підходах до генерування і перевірки підписів. DSA базується на дискретному логарифмуванні, тоді як RSA використовує факторизацію великих чисел. Це впливає на розмір ключів і швидкість операцій. Зокрема, DSA зазвичай використовує менші ключі для забезпечення того ж рівня безпеки, що і RSA, але може бути менш ефективним у випадку підписання дуже великих обсягів даних.

Також DSA має особливий механізм підписання, який включає використання випадкових чисел. Це підвищує рівень безпеки, оскільки для кожного підпису генерується унікальне випадкове число, яке використовується разом із приватним ключем для створення підпису. Важливою особливістю є те, що DSA є стандартом, схваленим урядом США, і широко використовується у різних стандартах та протоколах безпеки, таких як DSS (Digital Signature Standard).

Алгоритм DSA передбачає певні математичні обчислення, які включають генерацію випадкового числа, обчислення його модульного експонування, обчислення хешу повідомлення та використання приватного ключа для створення підпису. Верифікація підпису полягає у порівнянні обчислених значень з підписаними даними, що дозволяє перевірити автентичність і цілісність повідомлення. В цілому, DSA є надійним і ефективним інструментом для цифрової підписки, який забезпечує високий рівень безпеки та аутентифікації повідомлень.

Математичні основи алгоритму цифрового підпису DSA базуються на концепціях з теорії чисел, зокрема на дискретному логарифмуванні. Алгоритм використовує три великі числа: просте число p, його простий дільник q і число g, яке є первісним коренем по модулю p.

Хеш у алгоритмах цифрового підпису виконує ключову роль у забезпеченні безпеки та ефективності процесу підписання. Він перетворює повідомлення будь-якої довжини на фіксовану довжину, що значно зменшує обсяг даних, які потрібно обробляти. Це забезпечує швидкість і зручність роботи з алгоритмом. Хеш-функція також гарантує, що навіть незначні зміни у вихідному повідомленні призведуть до істотно іншого хеш-значення, що дозволяє виявити будь-які спроби підробки або несанкціонованої модифікації даних. Використання хешу забезпечує цілісність повідомлення, оскільки підпис накладається на хеш, а не на саме повідомлення, що знижує ризик підміни або пошкодження даних під час передачі. Хеш-функція додає додатковий рівень безпеки, оскільки складність обчислення зворотного значення хешу робить важким відтворення оригінального повідомлення чи пошук іншого повідомлення з таким самим хешем. Це запобігає атакам з підбору, коли зловмисник намагається знайти інше повідомлення з тим самим цифровим підписом. В цілому, хеш-функція є важливим елементом алгоритмів цифрового підпису, який забезпечує ефективність, цілісність та безпеку процесу підписання та перевірки підписів.

Алгоритм цифрового підпису DSA складається з кількох основних етапів: генерація ключів, підписання повідомлення та верифікація підпису. Нижче наведено покроковий опис:

- Обираємо велике просте число p
- Обираємо просте число q яке є дільником $p-1$ (тобто $p-1=kq$ для деякого цілого числа k)
- Обираємо випадкове число g є первісним коренем по модулю p. Формули наведено нижче

$$g = h^{(p-1)/q} mod\ p, h\ \in \{2, \dots, p-2\} \tag{2.8}$$

– Обираємо випадкове число x (приватний ключ) в діапазоні $1 \leq x \leq q - 1$є
– Обчислюємо публічний ключ y за формулою (2.9)

$$y = g^x \, mod \, p \tag{2.9}$$

Далі будемо розглядати підписання повідомлення, алгоритм дій для цього процесу буде наведено нижче у послідовності виконання:

– Обчислюємо хеш повідомлення m за допомогою криптографічної хещ-функції H, а саме $H(m)$
– Обираємо випадкове число k в діапазоні $1 \leq k \leq q - 1$
– Обчислюємо значення r за формулою (2.10):

$$r = (g^k \, mod \, p) \, mod \, p \tag{2.10}$$

– Обчислюємо значення s за формулою (2.11)

$$S = k^{-1}(H(m) + xr) \, mod \, q \tag{2.11}$$

– Підпис повідомлення складається з пари чисел (r, s)

Сам алгоритм цілком зрозумілий, проте підписом не обмежується, як і будь який інший алгоритм підпису, далі ми розглянемо алгоритм верифікації цифрового підпису:

– Перевіряємо що $0 < r < q$ та $0 < s < q$. Якщо ці нерівності не справджуються, то підпис не дійсний.
– Обчислюємо хеш повідомлення $H(m)$
– Обчислюємо значення w за формулою (2.12)

$$w = s^{-1} \, mod \, q \tag{2.12}$$

– Обчислюємо значення u_1 *і* u_2 за формулами (2.13) та (2.14) відповідно

$$u_1 = (H(m) \times w)\, mod\, q \tag{2.13}$$

$$u_2 = (r \times w)\, mod\, q \tag{2.14}$$

– Обчислюємо значення v за формулою (2.15)

$$V = \left((g^{u_1} \times y^{u_2})\, mod\, p\right) mod\, q \tag{2.15}$$

– Якщо $v = r$ підпис дійсний

Цей покроковий алгоритм з формулами показує, як DSA використовує криптографічні принципи для забезпечення автентичності та цілісності повідомлень. Схематичне зображення алгоритму наведено на рисунку 2.2.

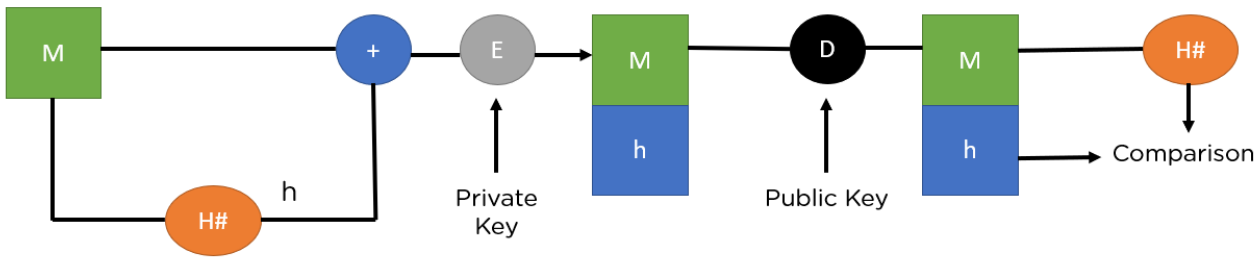

Рисунок 2.2 – Схема роботи алгоритму DSA

Переваги алгоритму DSA включають високий рівень безпеки, оскільки він базується на складності обчислення дискретного логарифму, що робить його стійким до криптографічних атак. Використання різних випадкових чисел для кожного підпису додає додатковий рівень захисту, запобігаючи повторному використанню підписів. DSA також є офіційним стандартом, схваленим урядом США, що підвищує його довіру та широке застосування у державних і комерційних структурах. Завдяки меншому розміру ключів порівняно з RSA, він

забезпечує той самий рівень безпеки при меншому обсязі даних, що може бути перевагою в системах з обмеженими ресурсами.

Недоліки алгоритму DSA включають його відносну повільність при підписанні повідомлень, особливо у порівнянні з іншими алгоритмами, такими як RSA. Це може бути критичним у системах, де потрібна висока продуктивність і швидка обробка великої кількості підписів. Вибір випадкового числа k є вразливим місцем: якщо k буде згенеровано неправильно або виявиться, це може призвести до компрометації приватного ключа. Крім того, DSA не може бути використаний для шифрування даних, а лише для підпису і перевірки підпису, що обмежує його універсальність у порівнянні з іншими алгоритмами, такими як RSA, які можуть виконувати обидві функції.

Алгоритм цифрового підпису DSA, розроблений NIST, забезпечує високий рівень безпеки завдяки використанню дискретного логарифмування та випадкових чисел. Він ефективний у використанні ключів і є офіційним стандартом, але має недоліки, такі як повільність підписання, вразливість при неправильному генеруванні випадкових чисел та обмежена функціональність без підтримки шифрування даних.

2.3 Алгоритм цифрового підпису ECDSA

Еліптична криптографія з підписом (ECDSA) стала важливим напрямком у сучасній криптографії, виправдавши очікування щодо стійкості та ефективності у захисті інформації. Ця галузь криптографії базується на застосуванні еліптичних кривих над скінченними полями і використовується для створення безпечних протоколів цифрового підпису.

Розвиток ECDSA визначений відкриттям потенційної стійкості еліптичних кривих для криптографічних цілей. Згадані криві над скінченними полями стали обєктом інтенсивних математичних досліджень в другій половині XX століття, коли вчені, такі як Джон Тейтл та Артур Кейслер, виявили їхню властивість у захисті інформації. Виділення еліптичних кривих у якості потенційно важливого

обєкта для криптографічних операцій відкрило шлях для подальших досліджень у напрямку розробки ефективних криптографічних протоколів.

В середині 1980-х років еліптична криптографія набула практичного застосування завдяки роботі таких вчених, як Ніл Фергюсон, Вільям Сталл та Джон Гордон. Вони розробили ECDSA, алгоритм цифрового підпису, що базується на властивостях еліптичних кривих. Цей алгоритм відкрив нові перспективи у криптографії та забезпечив основу для подальшого розвитку та стандартизації в галузі цифрового підпису.

ECDSA здобула визнання в криптографічній громаді та стала важливим інструментом для забезпечення безпеки в інформаційних системах. Її використання в різних галузях, таких як фінанси, мережеві технології та електронна комерція, підкреслює її значення як надійного та ефективного засобу захисту конфіденційної інформації.

Еліптична криптографія з підписом (ECDSA) базується на математичних властивостях еліптичних кривих над скінченними полями. Принцип роботи цього алгоритму полягає в створенні цифрового підпису для повідомлення та перевірці цього підпису для підтвердження автентичності повідомлення.

Спершу, вибираються параметри еліптичної кривої над скінченним полем, такі як коефіцієнти a та b і модуль p. Після вибору параметрів вибираються точки на кривій, включаючи точку бази, що використовується для генерації ключів.

Для створення підпису використовується приватний ключ, який є великим випадковим числом. З цього приватного ключа обчислюється відповідний публічний ключ, що є точкою на еліптичній кривій. Потім, для підпису повідомлення обчислюються числа r та s, які є частинами підпису.

У перевірці підпису використовується публічний ключ, який вже відомий отримувачеві. Вона включає обчислення хеша повідомлення та подальшу перевірку підпису, в якій порівнюються обчислені значення r та s з відомими значеннями підпису.

Ефективність ECDSA полягає у використанні властивостей еліптичних кривих для забезпечення стійкості та ефективності алгоритму, а також у

можливості генерації коротких підписів, що зменшує обсяг даних, необхідних для передачі.

Алгоритм підпису за допомогою еліптичних кривих (ECDSA) є ключовим елементом еліптичної криптографії і використовується для забезпечення цілісності, аутентичності та невідмінності даних. Принцип його роботи полягає у використанні приватного ключа для створення цифрового підпису, який потім може бути перевірений за допомогою відповідного публічного ключа.

Основні кроки алгоритму підпису ECDSA наступні:

– Спершу генерується приватний ключ d, який є випадковим числом у заданому діапазоні. Публічний ключ Q обчислюється шляхом множення точки бази (генератора) еліптичної кривої на приватний ключ за формулою (2.16), у якій G – це точка бази, а × — операція множення точки на скаляр.

$$Q = d \times G \tag{2.16}$$

– Генерується випадкове число k, яке є великим цілим числом та знаходиться у заданому діапазоні.

– Обчислюється точка (x_1, y_1)на еліптичній кривій, яка є результатом множення точки бази на k за формулою (2.17)

$$(x_1, y_1) = k \times G \tag{2.17}$$

– Обчислюється координата r, яка є x-координатою точки (x_1, y_1) за формулою (2.18), у якій n — порядок групи точок на еліптичній кривій.

$$r = x_1 \, mod \, n \tag{2.18}$$

– Обчислюємо значення s яке є результатом обчислення за формулою (2.19), де $H(m)$ – хеш повідомлення m, k^{-1} – обернене до k по модулю n

$$s = k^{-1}(H(m) + d \cdot r) \, mod \, n \tag{2.19}$$

– Підписом повідомлення є пара чисел (r, s).

Цей підпис може бути перевірений за допомогою відповідного публічного ключа та оригінального повідомлення. Перевірка підпису виконується за наступними кроками:

– Отримання публічного ключа Q та значень (r, s)підпису.

– Обчислення хешу повідомлення $H(m)$.

– Обчислення оберненого до s по модулю n

– Обчислення двох елементів u_1 та u_2 за формулами (2.20) та (2.21) відповідно

$$u_1 = H(m) \cdot s^{-1} \, mod \, n \tag{2.20}$$

$$u_2 = r \cdot s^{-1} \, mod \, n \tag{2.21}$$

– Обчислення точки (x_1, y_1)на еліптичній криві за формулою (2.22)

$$(x_1, y_1) = u_1 \times G + u_2 \times Q \tag{2.22}$$

– Перевірка умови $r = x_1 \, mod \, n$. Якщо умова виконується, то підпис дійсний.

Сематичне зображення роботи цього алгоритму наведено на рисунку 2.3.

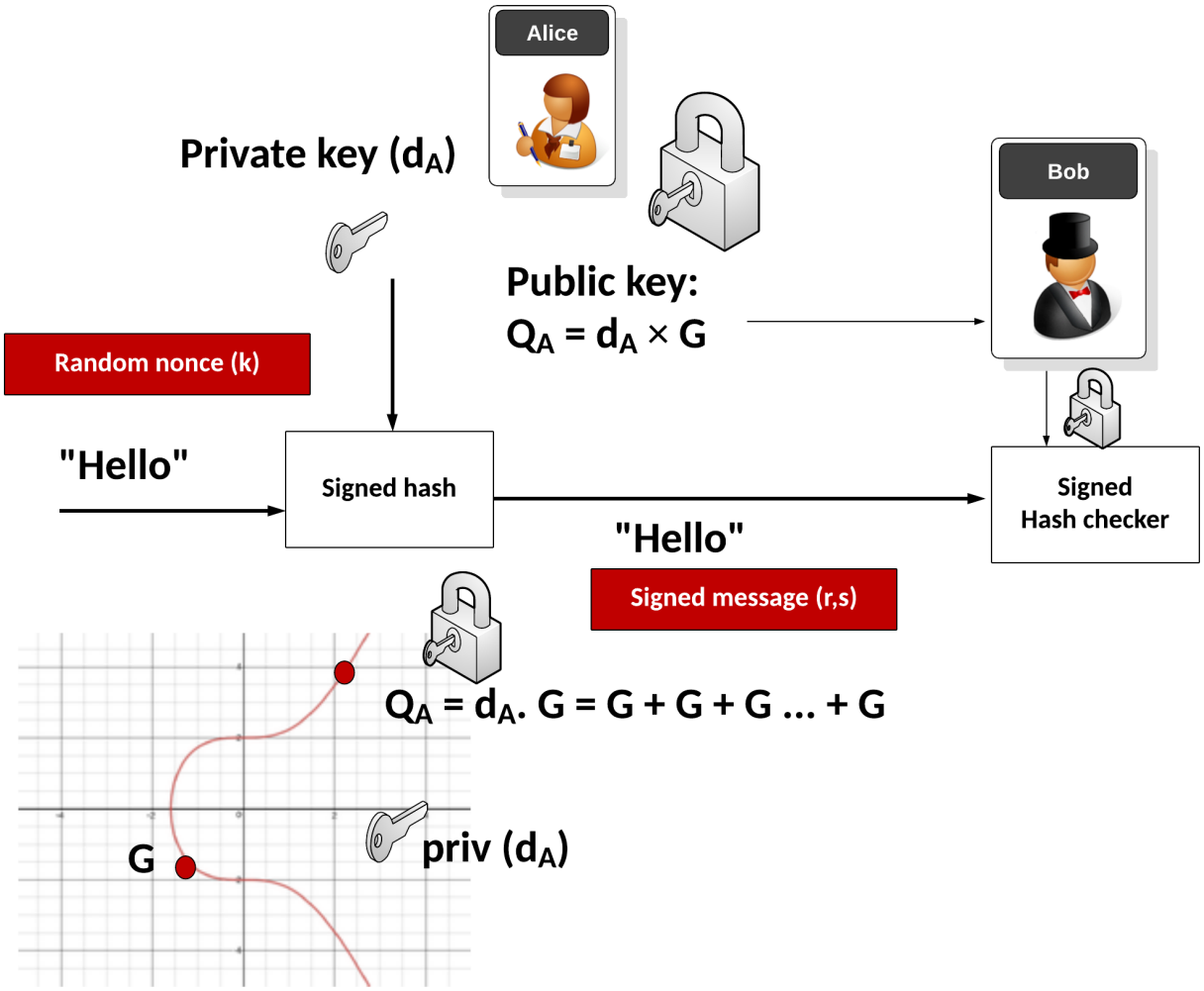

Рисунок 2.3 – Схематичне зображення алгоритму ECDSA

Алгоритм ECDSA є ефективним та широко використовуваним методом для забезпечення цілісності та автентичності даних у цифрових системах. Він володіє високою стійкістю до криптографічних атак та може бути успішно застосований у різних галузях, включаючи фінанси та мережеві технології. Незважаючи на це, важливо враховувати потенційні ризики, повязані з вибором параметрів кривої та управлінням ключами, для забезпечення безпеки системи.

(

3 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АСИМЕТРИЧНИХ АЛГОРИТМІВ ШИФРУВАННЯ

3.1 Вибір середовища розробки програми та технічні характеристики пристроїв для аналізу алгоритмів шифрування

Для реалізації алгоритмів та дослідження їхньої ефективності було обрано мову програмування Python та інтегроване середовище розробки PyCharm. Python є високорівневою мовою програмування, відомою своєю простотою і зрозумілим синтаксисом. Створена Гвідо ван Россумом та випущена у 1991 році, Python швидко здобула популярність завдяки своїй потужності для досвідчених програмістів та легкості у використанні для новачків. Python підтримує різні парадигми програмування, такі як обєктно-орієнтоване, функціональне та імперативне програмування.

Перед вибором було проведено дослідження ринку та виявлено, що саме ця мова є одним з фаворитів для розробки під різні мобільні та десктопні платформи. Перелгянути статистику можна на рисунках 3.1 – 3.3

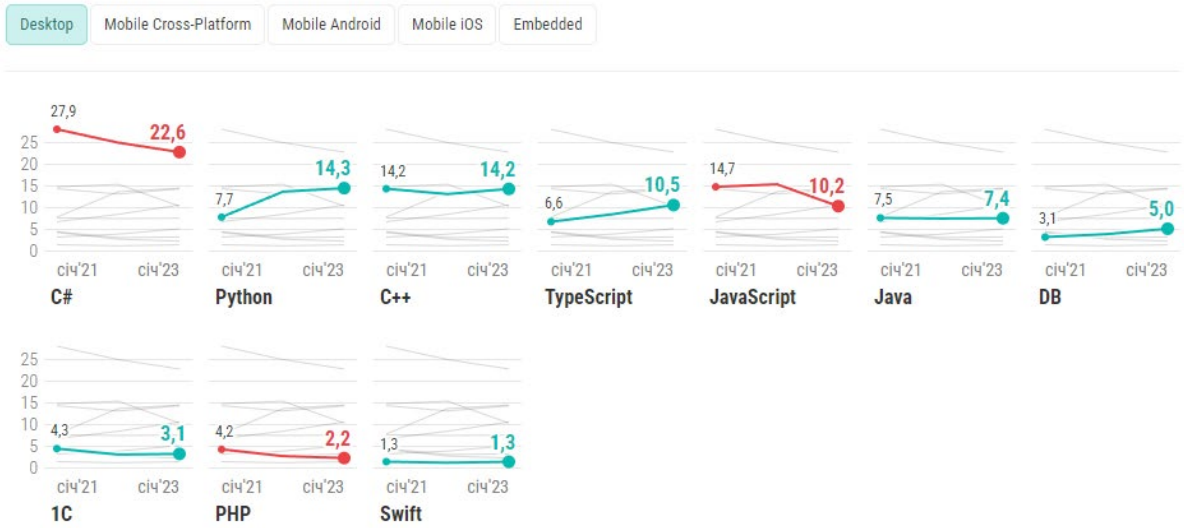

Рисунок 3.1 – Розподіл мов програмування, що використовуються для розробки ПК ПЗ.

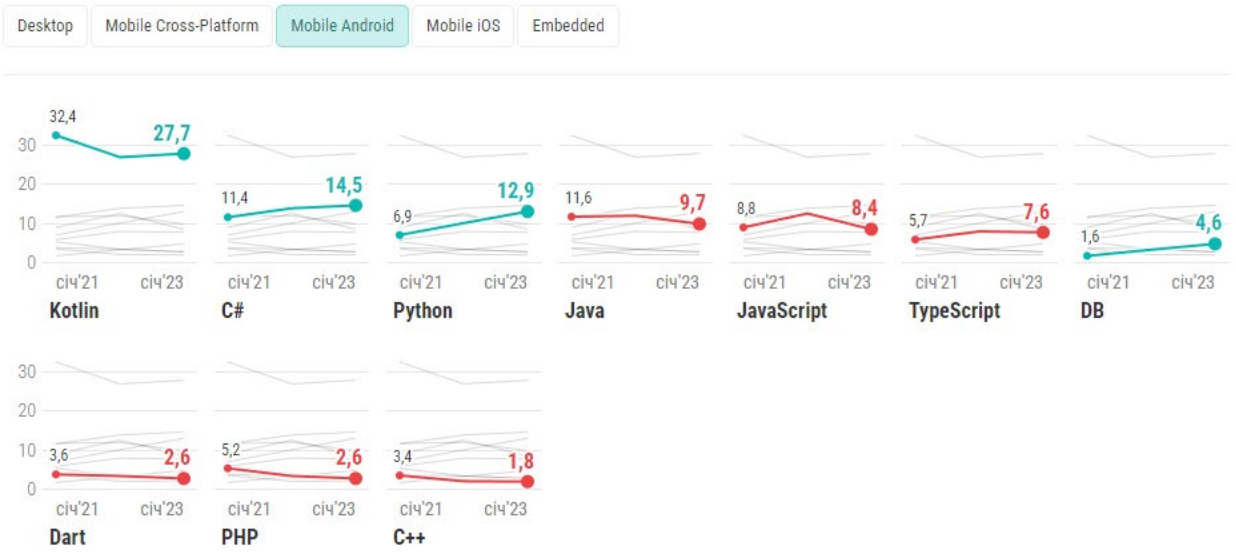

Рисунок 3.2 – Розподіл мов програмування, що використовуються для розробки Android ПЗ.

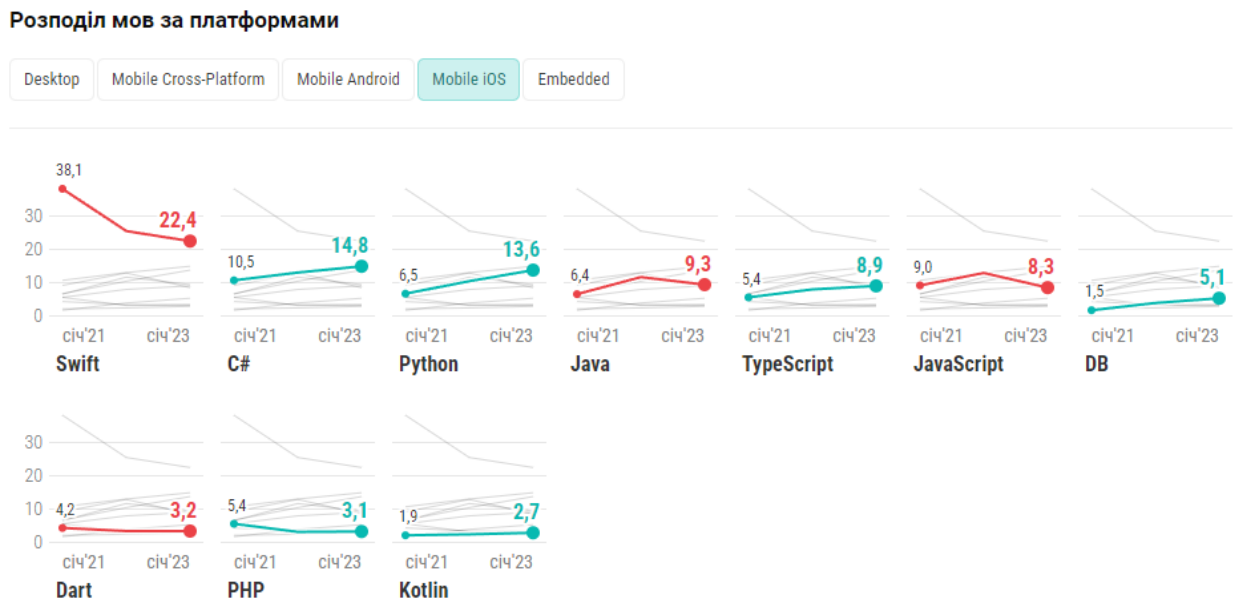

Рисунок 3.3 – Розподіл мов програмування, що використовуються для розробки IOS ПЗ.

Python є оптимальним вибором для виконання криптографічних завдань завдяки своїй простоті, розвиненій екосистемі бібліотек та активній спільноті розробників. Висока читабельність коду сприяє спрощенню процесів розробки та обслуговування криптографічних рішень, що є надзвичайно важливим у цій складній і відповідальній сфері. Потужні бібліотеки, такі як PyCryptodome та

cryptography, надають широкий спектр готових до використання алгоритмів шифрування, хешування та інших криптографічних операцій, що значно спрощує розробку безпечних додатків. Крім того, Python легко інтегрується з іншими мовами програмування, що дозволяє використовувати його в багатокомпонентних системах. Сильна підтримка спільноти забезпечує наявність численних ресурсів, прикладів коду та обговорень, що допомагає розробникам швидко знаходити рішення для своїх завдань.

Для отримання об'єктивних і репрезентативних результатів необхідно обрати методи оцінки ефективності алгоритмів цифрового підпису. У цій роботі для оцінки були обрані такі метрики: час виконання, використання CPU та використання пам'яті.

Час виконання є критичною метрикою для алгоритмів цифрового підпису, оскільки швидкість підписання та перевірки підписів безпосередньо впливає на загальну продуктивність системи. Тривалий процес підписання або перевірки може уповільнити передачу даних або реакцію системи. Крім того, час виконання може впливати на безпеку: якщо він значно змінюється залежно від вхідних даних, зловмисники можуть використовувати цю інформацію для аналізу підписів, що відоме як атака за часом. Обсяг даних також має значення: чим більше даних потрібно підписати або перевірити, тим більше часу це займає. Це особливо важливо для великих обсягів даних, як у хмарних обчисленнях. Час виконання також може впливати на тривалість ефективності ключа підпису: якщо процес підписання займає багато часу, ключ може стати застарілим до завершення процесу. Таким чином, оптимізація часу виконання є важливою для розробки ефективних і безпечних систем цифрового підпису.

Використання CPU також є важливою метрикою, оскільки системи з цифровим підписом не завжди мають високу продуктивність. Велике навантаження може спричинити незручності або навіть відмову системи. Наприклад, у реальних часах перевірка підписів повинна бути швидкою, щоб не створювати затримок у роботі системи.

Використання пам'яті є важливою метрикою для алгоритмів цифрового підпису, оскільки багато систем не мають великого запасу оперативної пам'яті,

але повинні виконувати багато паралельних завдань. Тому мінімізація використання пам'яті є необхідною для збереження продуктивності. Це особливо актуально для мобільних пристроїв і вбудованих систем, де ресурси пам'яті обмежені. Ефективне використання пам'яті дозволяє підтримувати стабільну роботу системи навіть при обробці великої кількості підписів.

Таким чином, оцінка ефективності алгоритмів цифрового підпису на основі часу виконання, використання CPU та використання пам'яті є ключовим аспектом для забезпечення високої продуктивності та надійності криптографічних систем.

3.2 Розробка програми, що реалізує досліджуванні алгоритми шифрування та вимірює ефективність алгоритмів

PyCryptodome — це потужна бібліотека для Python, призначена для криптографії. Вона є форком бібліотеки PyCrypto і містить численні вдосконалення та нові функції. PyCryptodome забезпечує реалізацію різних криптографічних примітивів, таких як симетричні шифри (AES, DES), асиметричні шифри (RSA, DSA), хеш-функції (SHA, MD5) та інші криптографічні алгоритми. Бібліотека підтримує різні операції, включаючи шифрування/дешифрування, підписування, верифікацію та генерацію ключів. Вона легко інтегрується з Python і має гарну документацію, що робить її зручною для використання в різних криптографічних завданнях.

Timeit — це бібліотека призначена для вимірювання часу виконання невеликих фрагментів коду. Вона забезпечує точність завдяки уникненню впливу інших процесів системи. Основні методи бібліотеки дозволяють швидко оцінити продуктивність коду, порівнюючи різні варіанти реалізації.

SymPy — це бібліотека для символьних обчислень у Python, яка дозволяє виконувати алгебраїчні перетворення, розвязувати рівняння, обчислювати границі, похідні та інтеграли, а також працювати з матрицями та диференціальними рівняннями. Вона забезпечує зручні інструменти для

маніпуляції математичними виразами та їхньої автоматизації, що робить її корисною для наукових досліджень, інженерних розрахунків та навчання математики.

Оглянемо функції що використовуються для реалізації алгоритму шифрування та дешифрування RSA:

Перша функція яку ми розглянемо генерує параметри для шифрування, а саме p, q, n а також обчислює функцію Ейлера $\varphi(n)$, саму функцію описано в Лістингу 3.1.

Лістинг 3.1 – Код функції для генерації параметрів RSA

```
def generate_keypair(length):
  p = number.getPrime(length)
  q = number.getPrime(length)
  n = p * q
  phi = (p - 1) * (q - 1)

  # Choose an integer e such that e and phi(n) are coprime
  e = random.randrange(2, phi)
  g = gcd(e, phi)
  while g != 1:
    e = random.randrange(2, phi)
    g = gcd(e, phi)
  # Calculate d
  d = mod_inverse(e, phi)

  # Public key (e, n) and private key (d, n)
  return ((e, n), (d, n))
```

Наступна функція яку ми розглянемо ця функція gcd, вона обчислює найбільший спільний дільник (НСД) двох чисел за допомогою алгоритму Евкліда. Вона приймає два аргументи a і b. В циклі while, поки b не дорівнює нулю, змінні a і b замінюються на b і залишок від ділення a на b, відповідно. Після завершення циклу функція повертає значення a, яке є НСД. Код функції на мові програмування Python наведено у лістингу 3.3.

Лістинг 3.2 – Код функції для розширеного алгоритму Евкліда

```
def gcd(a, b):
  while b != 0:
    a, b = b, a % b
  return a
```

Функція sign_message реалізує процес цифрового підпису повідомлення за допомогою алгоритму RSA. Вона приймає два аргументи: закритий ключ і повідомлення. Закритий ключ представлений у вигляді кортежу (d, n), де d – приватний експонент, а n – модуль RSA. Повідомлення є рядком символів.

У функції спочатку здійснюється перетворення кожного символу повідомлення у відповідне числове значення за допомогою функції ord, яка повертає Unicode код символу. Далі, для кожного числового значення символу обчислюється цифровий підпис шляхом піднесення цього числа до степеня d та взяття залишку від ділення на n. Цей процес здійснюється з використанням функції pow, яка дозволяє ефективно виконувати операцію піднесення до степеня за модулем. Підсумковий результат, який є списком підписаних числових значень, повертається як підпис повідомлення.

Таким чином, функція sign_message забезпечує створення цифрового підпису повідомлення, що дозволяє отримувачеві перевірити цілісність і автентичність повідомлення за допомогою відповідного відкритого ключа. Код функції на мові програмування Python наведено у лістингу 3.3

Лістинг 3.3 – Код функції для шифрування RSA

```
def sign_message(private_key, message):
  d, n = private_key
  # Convert the message to a number based on the character and sign
it using a^d mod n
  signature = [pow(ord(char), d, n) for char in message]
  return signature
```

Ця функція реалізує перевірку цифрового підпису повідомлення, використовуючи алгоритм RSA. Основними компонентами RSA є відкритий ключ, що складається з двох чисел

e та онуки n, повідомлення, що потребує перевірки, та підпису. Підпис, який перевіряється, є результатом підписання оригінального повідомлення з використанням закритого ключа відправника.

Процес перевірки починається з дешифрування підпису, що виконується за допомогою відкритого ключа. Функція обчислює степінь кожного символу підпису за модулем n з експонентою e. Це здійснюється в одному рядку коду за допомогою виразу pow(char, e, n), де char представляє числове значення кожного символу підпису. Результатом є числова послідовність, яка далі перетворюється на символьну, утворюючи дешифроване повідомлення.

Завершальним етапом є порівняння отриманого дешифрованого повідомлення з оригінальним повідомленням. Якщо вони збігаються, це означає, що підпис дійсно належить повідомленню, і функція повертає істину, підтверджуючи автентичність підпису. Інакше повертається хиба, вказуючи на те, що підпис не є автентичним. Цей метод забезпечує надійну перевірку підпису завдяки криптографічним властивостям RSA, що робить його важливим інструментом для захисту даних та підтвердження їхньої цілісності. Код функції на мові програмування Python наведено у лістингу 3.4

Лістинг 3.4 – Код функції для дешифрування RSA

```
def verify_signature(public_key, message, signature):
  e, n = public_key
  # Verify the signature by converting it back to the original message
  decrypted_message = ''.join([chr(pow(char, e, n)) for char in signature])
  return decrypted_message == message
```

Функція main_rsa() реалізує механізм цифрового підпису та його перевірку в криптосистемі RSA з використанням ключової довжини 512 біт. На початку виконання, функція генерує пару ключів RSA (публічний і приватний ключ) за допомогою функції generate_keypair().

Далі, створюється повідомлення, яке необхідно підписати. Використовуючи приватний ключ, функція створює цифровий підпис повідомлення за допомогою функції sign_message(). Після цього цифровий підпис перевіряється за допомогою публічного ключа і функції verify_signature().

Функція повертає результат перевірки, який підтверджує автентичність та цілісність підписаного повідомлення. Завдяки цьому, main_rsa() демонструє

основний принцип роботи цифрового підпису в RSA, забезпечуючи криптографічний захист даних шляхом підтвердження, що повідомлення не було змінене і походить від надійного джерела.

Лістинг 3.5 – Код функції для шифрування та дешифрування RSA

```
def main_rsa():
  length = 512 # key length
  public_key, private_key = generate_keypair(length)

  message = "This is a secret message."
  signature = sign_message(private_key, message)
  verification = verify_signature(public_key, message,
signature)
  return verification
```

Далі будемо розглядати код програмної реалізації DSA. У цьому модулі функцій буде дещо менше функцій оскільки більшість формул цього підпису можуть бути реалізовані за допомогою вже існуючих модулів, або базових модулів Python.

Функція sign реалізує процес створення цифрового підпису за алгоритмом DSA, використовуючи параметри p, q, g та секретний ключ x. Спочатку генерується випадкове число k у діапазоні від 1 до q-1, яке слугуватиме для забезпечення випадковості підпису. Далі обчислюється значення r як залишок від ділення числа g, піднесеного до степеня k, за модулем p на q. Це забезпечує, що r завжди знаходиться в межах від 0 до q-1.

Для підпису повідомлення необхідно обчислити його геш-значення. Функція використовує алгоритм SHA-1 для створення гешу повідомлення, який потім перетворюється в ціле число. Це число буде використовуватись на наступних етапах для обчислення другого параметру підпису sign. Код функції наведено у лістингу 3.6

Лістинг 3.6 – Код функції для підписування повідомлення підписом DSA

```
def sign(message, p, q, g, x):
  k = random.randint(1, q-1)
  r = pow(g, k, p) % q
  hash_obj = hashlib.sha1()
  hash_obj.update(message.encode())
  hash_value = int.from_bytes(hash_obj.digest(), big)
```

```
  # Calculate s
  k_inv = pow(k, -1, q)
  sing = (k_inv * (hash_value + x * r)) % q

  return r, sing
```

Наступна функція реалізує перевірку цифрового підпису DSA (Digital Signature Algorithm). Вона приймає такі параметри: повідомлення, підписи r і s, а також параметри DSA: p, q, g і y. Функція перевіряє, чи дійсні значення r і s, які повинні бути в діапазоні від 0 до q. Якщо вони не відповідають цим умовам, функція повертає False.

Спочатку функція обчислює хеш повідомлення за допомогою SHA-1 і перетворює його в ціле число. Потім вона обчислює величину w, яка є оберненим значенням s за модулем q. Далі обчислюються значення u1 і u2, які залежать від хеш-значення повідомлення, r і w. Потім обчислюється величина v, яка є модульним добутком g в степені u1 і y в степені u2, взятим за модулем p, і знову за модулем q.

Фінальна перевірка полягає в порівнянні v з r. Якщо вони рівні, підпис вважається дійсним і функція повертає True. Якщо ні, то повертається False. Код функції наведено у лістингу 3.7

Лістинг 3.7 – Код функції для перевірки підпису повідомлення

```
def verify(message, r, s, p, q, g, y):
  if not (0 < r < q) or not (0 < s < q):
    return False

  # Calculate hash of the message
  hash_obj = hashlib.sha1()
  hash_obj.update(message.encode())
  hash_value = int.from_bytes(hash_obj.digest(), big)
  # Calculate w
  w = pow(s, -1, q)
  # Calculate u1 and u2
  u1 = (hash_value * w) % q
  u2 = (r * w) % q

  # Calculate v
  v = ((pow(g, u1, p) * pow(y, u2, p)) % p) % q

  return v == r
```

Наступна функція реалізує генерацію ключів та підписання повідомлення за допомогою алгоритму DSA. Спочатку генеруються прості числа p і q, де p-1 кратне q. Потім обчислюється значення g як степінь випадкового числа h по модулю p. Приватний ключ x є випадковим числом між 1 і q-1, а публічний ключ y обчислюється як g в степені x по модулю p.

Функція підписує повідомлення, використовуючи функцію sign, яка повертає підписи r і s. Після цього перевіряє підпис за допомогою функції verify, яка використовує початкові параметри та публічний ключ y. Результат перевірки повертається функцією main_dsa: True, якщо підпис дійсний, інакше False. Код функції наведено нижче у лістингу 3.8

Лістинг 3.8 – Код функції для перевірки підпису повідомлення

```
def main_dsa():
  while True:
    p = number.getPrime(512)
    q = number.getPrime(8)
    if (p - 1) % q == 0:
      break
  # Generate g such that 1 < g < p
  h = random.randint(2, p-2)
  g = pow(h, (p-1)//q, p)
  # Generate private key
  x = random.randint(1, q-1)
  # Generate public key
  y = pow(g, x, p)
  message = "This is a secret message."
  # Signing the message
  r, s = sign(message, p, q, g, x)
  # Verifying the signature
  is_valid = verify(message, r, s, p, q, g, y)
  return is_valid
```

Далі розглянемо алгоритм цифрового підпису на основі еліптичних кривих. Параметрами кривої було обрано досить поширені secp256k1. secp256k1 — це крива Еліптичної кривої, яка широко використовується в криптографії. Вона належить до стандартів, визначених NIST (Національний інститут стандартів і технологій США) для використання в алгоритмах на основі еліптичних кривих.

Параметри кривої були взяті із сайту NIST та наведені у лістингу 3.9.

Лістинг 3.9 – Параметри еліптичної кривої

```
    # Elliptic curve parameters (secp256k1)
    p=0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F
    a = 0
    b = 7
    Gx=0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798
    Gy=0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8
    n=0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141
```

Далі розглянемо функцію перевірки is_on_curve, вонаперевіряє, чи лежить точка на еліптичній кривій в рамках алгоритму ECDSA. Вона приймає точку як аргумент і повертає True, якщо точка належить кривій, і False в іншому випадку. Якщо аргументом є None, функція повертає True, оскільки None вважається точкою на нескінченності, яка завжди належить кривій.

У випадку, коли точка не є None, вона розкладається на координати x і y. Функція перевіряє рівність (y ** 2 - x ** 3 - a * x - b) % p == 0, де a, b, і p є параметрами еліптичної кривої. Якщо рівність виконується, це означає, що точка задовольняє рівнянню кривої і, отже, належить їй. В іншому випадку точка не лежить на кривій. Код функції наведено у лістингу 3.10.

Лістинг 3.10 – Код функції для перевірки належності точки до кривої

```
def is_on_curve(point):
  """Check if a point is on the elliptic curve."""
  if point is None:
    return True
  x, y = point
  return (y ** 2 - x ** 3 - a * x - b) % p == 0
```

Функція point_add реалізує операцію додавання двох точок на еліптичній кривій. Вона приймає два аргументи point1 і point2, кожен з яких є кортежем координат (x, y). Якщо перша точка дорівнює None, функція повертає другу точку, і навпаки. Це обробляє випадки, коли одна з точок є точкою на нескінченності.

Функція спочатку витягує координати x і y з обох точок. Якщо абсциси точок однакові, а ординати різні, це означає, що точки є однаковими за x, але

знаходяться на протилежних сторонах осі x, і їх сума дорівнює точці на нескінченності, що повертає None.

Якщо абсциси і ординати обох точок однакові, функція використовує формулу для додавання точки до самої себе на еліптичній кривій. Вона обчислює нахил дотичної до кривої в цій точці, використовуючи формулу (3 * x1 ** 2 + a) * mod_inverse(2 * y1, p), де a – це параметр кривої, а mod_inverse – функція, що обчислює обернене значення за модулем p.

Якщо точки різні, функція обчислює нахил прямої, що проходить через ці точки, за формулою (y2 - y1) * mod_inverse(x2 - x1, p). Потім обчислюються координати нової точки на кривій. Абсциса нової точки дорівнює m ** 2 - x1 - x2, а ордината обчислюється як y1 + m * (x3 - x1). Остаточні координати нової точки обчислюються за модулем p, щоб повернути результат у полі цілих чисел за модулем.

Таким чином, функція повертає нову точку на еліптичній кривій як кортеж (x3 % p, -y3 % p). Код програмної реалізації на мові програмування Python наведено нижче у лістингу 3.11.

Лістинг 3.11 – Код функції для додавання точки

```
def point_add(point1, point2):
  """Add two points on the elliptic curve."""
  if point1 is None:
    return point2
  if point2 is None:
    return point1
  x1, y1 = point1
  x2, y2 = point2
  if x1 == x2 and y1 != y2:
    return None
  if x1 == x2:
    m = (3 * x1 ** 2 + a) * mod_inverse(2 * y1, p)
  else:
    m = (y2 - y1) * mod_inverse(x2 - x1, p)
  x3 = m ** 2 - x1 - x2
  y3 = y1 + m * (x3 - x1)
  return (x3 % p, -y3 % p)
```

Функція scalar_mult виконує множення точки на еліптичній кривій на ціле число. Вона приймає два аргументи: ціле число k і точку point. Спочатку перевіряється, чи кратне k порядку групи n, або чи є точка None, у такому разі

функція повертає None. Якщо k негативне, функція рекурсивно викликає саму себе з протилежним значенням k і інверсією координат точки. Ініціалізуються змінні result і addend, де result спочатку None, а addend дорівнює початковій точці. У циклі перевіряється останній біт k. Якщо він дорівнює одиниці, до result додається addend за допомогою функції point_add. Потім addend подвоюється, знову використовуючи point_add, і k зсувається вправо на один біт. Процес повторюється, поки k не стане нулем. В результаті функція повертає точку, яка є результатом множення. Код функції наведено у лістингу 3.12.

Лістинг 3.12 – Код функції множення точки на скаляр

```
def scalar_mult(k, point):
  if k % n == 0 or point is None:
    return None
  if k < 0:
    return scalar_mult(-k, (-point[0], -point[1]))
  result = None
  addend = point
  while k:
    if k & 1:
      result = point_add(result, addend)
    addend = point_add(addend, addend)
    k >>= 1
  return result
```

Функція generate_keypair() є частиною програмної реалізації алгоритму ECDSA, що генерує пару ключів: приватний та публічний. Спочатку випадковим чином вибирається приватний ключ d, що є цілим числом у діапазоні від 1 до n-1, де n – це порядок генератора кривої. Після цього обчислюється публічний ключ Q за допомогою скалярного множення приватного ключа d на базову точку кривої з координатами (Gx, Gy). Потім перевіряється, чи належить отримана точка кривій, використовуючи функцію is_on_curve(). Якщо точка не належить кривій, функція викидає помилку з відповідним повідомленням. У результаті функція повертає пару ключів: приватний ключ d та публічний ключ Q. Код функції наведено у лістингу 3.13.

Лістинг 3.13 – Код функції генерації ключів

```
def generate_keypair():
  d = random.randrange(1, n)
  Q = scalar_mult(d, (Gx, Gy))
```

Продовження лістингу 3.13

```
    assert is_on_curve(Q), "Public key is not on the curve."
    return d, Q
```

Функція sign_message реалізує процес підпису повідомлення за допомогою алгоритму ECDSA (Elliptic Curve Digital Signature Algorithm). Вона приймає два параметри: закритий ключ та саме повідомлення. Спочатку повідомлення хешується за допомогою SHA-256, і результат перетворюється у велике ціле число. Потім ініціалізуються змінні r та s, які будуть містити підпис.

Для генерації підпису функція входить у цикл, який триває, доки r або s дорівнюють нулю. Усередині циклу спочатку генерується випадкове число k у діапазоні від 1 до n-1, де n – порядок основної точки еліптичної кривої. Потім обчислюються координати точки, що дорівнює добутку числа k на базову точку кривої (Gx, Gy). Значення r визначається як координата x отриманої точки, взята за модулем n. Значення s обчислюється за формулою, що включає обернене до k за модулем n, хеш повідомлення z, значення r та закритий ключ.

Цикл повторюється доти, доки значення r та s не будуть відмінними від нуля, що забезпечує коректність підпису. Після завершення циклу функція повертає пару чисел r і s, які є цифровим підписом повідомлення. Реалізацію функції наведено у лістингу 3.14

Лістинг 3.14 – Код функції підпису повідомлення

```
def sign_message(private_key, message):
  z = int.from_bytes(hashlib.sha256(message).digest(), big)
  r, s = 0, 0
  while r == 0 or s == 0:
    k = random.randrange(1, n)
    x, y = scalar_mult(k, (Gx, Gy))
    r = x % n
    s = (mod_inverse(k, n) * (z + r * private_key)) % n
  return r, s
```

Функція verify_signature реалізує перевірку підпису за допомогою алгоритму ECDSA. Вона приймає три параметри: відкритий ключ, повідомлення і підпис, який складається з двох частин, r та s. Спершу функція перевіряє, чи

лежать r та s у допустимому діапазоні значень, що менше деякого числа n. Якщо ці умови не виконуються, повертається False.

Далі функція обчислює хеш повідомлення за допомогою SHA-256 і перетворює його на ціле число. Після цього обчислюється обернене значення s за модулем n, що використовується для обчислення значень u1 та u2. Ці значення потрібні для подальших обчислень координат точки на кривій. Вони визначаються як результат додавання двох точок на кривій, кожна з яких отримується множенням відповідного скаляра на базову точку кривої та відкритий ключ.

Отримана точка перевіряється на належність до кривої, і якщо вона не лежить на ній, виводиться відповідне повідомлення. Нарешті, функція порівнює значення r з координатою x обчисленої точки за модулем n і повертає True або False залежно від результату цього порівняння. Код функції наведено у лістингу 3.15.

Лістинг 3.15 – Код функції перевірки підпису повідомлення

```
def verify_signature(public_key, message, signature):
  r, s = signature
  if not (1 <= r < n and 1 <= s < n):
    return False
  z = int.from_bytes(hashlib.sha256(message).digest(), big)
  w = mod_inverse(s, n)
  u1 = (z * w) % n
  u2 = (r * w) % n
  x, y = point_add(scalar_mult(u1, (Gx, Gy)), scalar_mult(u2,
public_key))
  if not is_on_curve((x, y)):
    print("Computed point is not on the curve.")
  return (r % n) == (x % n)
```

Також у нас є модуль measure.py, який реалізує замір використання ресурсів системи та часу виконання. Алгоритм заміру використання ЦП, памяті та часу виконання у цьому коді реалізовано через функцію measure_performance, яка приймає функцію func та її аргументи. Спершу фіксується час початку виконання функції за допомогою time.time().Щоб заміряти використання ЦП та памяті, створюється обєкт процесу за допомогою модуля psutil, і фіксується початковий відсоток використання ЦП та використання памяті.

Далі виконується передана функція з аргументами, а після її завершення знову фіксується час, щоб обчислити загальний час виконання. Потім вимірюється кінцевий відсоток використання ЦП, і на основі різниці між початковим та кінцевим показниками визначається загальне використання ЦП під час виконання функції.

На основі зібраних даних створюється словник performance_metrics, який містить результати виконання функції, час виконання, використання памяті, а також використання ЦП. Цей словник повертається як результат роботи функції measure_performance. Програмну реалізацію цієї функції наведено у лістингу 3.16.

Лістинг 3.16 – Код функції заміру споживання системних ресурсів

```
def measure_performance(func, *args):
  process = psutil.Process(os.getpid())

  start_memory = process.memory_info().rss
  start_cpu_percent = psutil.cpu_percent(interval=None)

  execution_time = timeit.timeit(lambda: func(*args), number=1)

  end_memory = process.memory_info().rss
  end_cpu_percent = psutil.cpu_percent(interval=None)

  avg_cpu_percent = (start_cpu_percent + end_cpu_percent) / 2
  memory_used = end_memory - start_memory
  performance_metrics = {
    "function_name": "",
    execution_time: execution_time,
    current_memory_usage: memory_used,
    cpu_usage: avg_cpu_percent,
  }
  return performance_metrics
```

3.3 Тестування та порівняння ефективності досліджуваних алгоритмів шифрування

Для оцінки ефективності досліджуваних алгоритмів цифрового підпису, а саме RSA, DSA, ECDSA, було проведено серію експериментів. Ми вимірювали час виконання, споживання процесорного часу, використання пам’яті та CPU для

кожного алгоритму за допомогою спеціально розробленої програми. Порівняльний аналіз є важливою частиною дослідження, оскільки він дозволяє розробникам і системним адміністраторам вибрати найкращий алгоритм цифрового підпису для забезпечення автентичності та цілісності даних. Наприклад, деякі алгоритми можуть забезпечувати вищий рівень безпеки за рахунок більшого використання ресурсів, тоді як інші можуть бути менш вимогливими до ресурсів, але забезпечувати нижчий рівень безпеки. Розуміння цих компромісів є ключовим для прийняття обґрунтованих рішень, що допоможуть забезпечити високий рівень захисту даних, мінімізуючи при цьому вплив на продуктивність системи.. Отже, результати виконання зображено на рисунку 3.4.

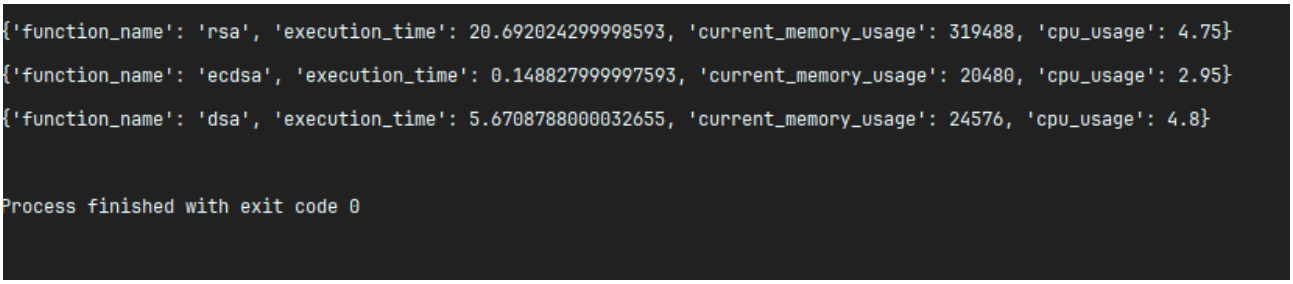

Рисунок 3.4 – Результат виконання програми алгоритмів цифрового підпису

З наведених даних можемо побачити що алгоритм RSA досить швидко виконується та використовує зовсім не багато ресурсів, а саме майже 4.65% процесора, що становить майже повністю завантажене одне ядро у нашій системі, та 319488байт пам’яті що у сучасних реаліях зовсім мало, проте дуже багато у порівнянні із конкурентами. Ба Також дуже великий час виконання 20,39 с.

Далі розглянемо ECDSA, який показав найкращі результати у виконанні, а саме 3.6% використання процесора, що складає дещо більше половини навантаження на одне ядро, 20480 байт пам’яті та дуже швидкий час виконання у 0,15 секунди.

У свою чергу алгоритм DSA показав дуже хороші результати, а саме 5 секунд виконання, при повністю навантаженому одному ядрі процесора (4,8%) та 24576 байт використання пам’яті. Такі показники можуть бути пов’язані із великою кількістю обчислень та перебором правильних параметрів для побудови криптосистеми.

Виходячи з наведених даних, можна зробити висновок, що ефективність сучасних асиметричних криптографічних алгоритмів значною мірою залежить від обчислювальних ресурсів і часу виконання. RSA, незважаючи на свою надійність, вимагає значних обсягів пам'яті і має тривалий час виконання, що може бути обмеженням для деяких застосувань. ECDSA демонструє найкращі показники завдяки низькому споживанню ресурсів і швидкому виконанню, що робить його ідеальним вибором для мобільних та інших ресурсомістких середовищ. DSA, хоча і показує хороші результати, все ж вимагає більше часу на виконання та дещо більше ресурсів у порівнянні з ECDSA, але може бути ефективним для застосувань, де критично важливі швидкість і обчислювальні можливості. Таким чином, вибір конкретного алгоритму залежить від специфічних вимог до безпеки, продуктивності та ресурсів у конкретній системі.

Результати тестів буде наведено у таблиці 3.1

Таблиця 3.1 – Результати показників ефективності алгоритмів хешування

Функція хешування	Час роботи (сек.)	Вико-ристання пам’яті (В)	Вико-ристання процесора (%)
RSA	20.69	319488	4.75
DSA	5.67	24576	4.80
ECDSA	0.15	20480	2.95

Алгоритм RSA є одним з найбільш відомих і широко використовуваних у світі. Його основними перевагами є висока надійність та стійкість до атак, що забезпечується завдяки складності факторизації великих чисел. Проте, його основний недолік полягає в досить великій довжині ключів, необхідних для досягнення високого рівня безпеки, що може знижувати швидкість роботи системи.

DSA, з іншого боку, має інший підхід до створення цифрових підписів, заснований на дискретному логарифмі. Цей алгоритм є менш вимогливим до обчислювальних ресурсів порівняно з RSA, але також вимагає довгих ключів для забезпечення достатнього рівня безпеки. DSA також підходить для застосувань, де важлива швидкість підписання даних.

ECDSA, що базується на еліптичних кривих, представляє собою сучасний і дуже ефективний підхід до асиметричної криптографії. Основною перевагою ECDSA є можливість забезпечення високого рівня безпеки при використанні значно коротших ключів, що робить його особливо привабливим для мобільних та інших ресурсомістких середовищ. Крім того, він забезпечує швидке створення та перевірку підписів, що є критичним у багатьох сучасних застосуваннях.

Загалом, кожен з розглянутих алгоритмів має свої унікальні переваги та недоліки, що визначає їхнє застосування в різних контекстах. RSA підходить для ситуацій, де необхідна максимальна надійність та стійкість до атак, DSA - для систем, де важлива швидкість та ефективність обчислень, а ECDSA - для мобільних та інших середовищ з обмеженими ресурсами. Вибір конкретного алгоритму залежить від специфічних вимог та умов застосування.

4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Правове забезпечення та організаційно-функціональна структура захисту населення.

«захист населення і територій від надзвичайних ситуацій техногенного та природного характеру - система організаційних, технічних, медико-біологічних, фінансово-економічних та інших заходів щодо запобігання та реагування на надзвичайні ситуації техногенного та природного характеру і ліквідації їх наслідків, що реалізуються центральними і місцевими органами виконавчої влади, органами місцевого самоврядування, відповідними силами та засобами підприємств, установ та організацій незалежно від форм власності і господарювання, добровільними формуваннями і спрямовані на захист населення і територій, а також матеріальних і культурних цінностей та довкілля;» [4]. Це включає правове забезпечення та організаційно-функціональну структуру, які працюють разом для ефективного реагування на будь-які загрози. Розглянемо ці аспекти детальніше.

Правове забезпечення захисту населення включає систему нормативно-правових актів, що регулюють діяльність у сфері цивільного захисту. Основні елементи правового забезпечення включають:

- Конституція України — гарантує право громадян на безпечне життя;
- Кодекс цивільного захисту України — основний документ, що визначає правові та організаційні засади цивільного захисту, функції та повноваження органів влади, підприємств, установ і організацій у сфері цивільного захисту;
- Закони України:

a) Закон України «Про правовий режим надзвичайного стану»;

b) Закон України «Про захист населення і територій від надзвичайних ситуацій техногенного та природного характеру»;

c) Закон України «Про національну безпеку України»;

– Постанови Кабінету Міністрів України — визначають порядок дій у разі виникнення надзвичайних ситуацій, регулюють створення та функціонування системи цивільного захисту;

– Накази та інструкції центральних органів виконавчої влади — регламентують конкретні аспекти діяльності у сфері цивільного захисту.

Організаційно-функціональна структура захисту населення передбачає наявність різних рівнів управління та координації дій. Основним елементом цієї структури на державному рівні є Державна служба України з надзвичайних ситуацій (ДСНС), яка є основним органом, відповідальним за реалізацію державної політики у сфері цивільного захисту. Здійснюючи свою діяльність ДСНС координує функціонування інших органів влади, підприємств та установ у разі НС.

На регіональному рівні створено обласні, міські та районні управління ДСНС, які забезпечують реалізацію заходів цивільного захисту на місцевому рівні та організовують і контролюють діяльність місцевих органів влади, підприємств та установ у разі НС.

Підприємства, установи та організації відповідальні за розробку та впровадження заходів з цивільного захисту на своїх обєктах на локальному рівні. Вони зобов’язані проводити навчання працівників, забезпечувати наявність необхідних засобів захисту та реагування на НС.

Громадські організації та добровільні формування діють на громадському рівні і можуть брати участь у заходах з цивільного захисту, проводити навчання та інформаційно-просвітницьку роботу серед населення.

Основні функції та завдання структури захисту населення включають:

- планування заходів цивільного захисту (розробка планів реагування на надзвичайні ситуації різних рівнів);

- попередження надзвичайних ситуацій (проведення моніторингу та оцінки ризиків, розробка заходів з їхнього зменшення);

- реагування на надзвичайні ситуації (організація та проведення рятувальних та інших невідкладних робіт у зоні НС);

- навчання та підготовка населення (проведення тренувань, навчань та інформування населення про дії у разі НС).

Ефективна система захисту населення потребує чіткої правової бази та організаційно-функціональної структури, що забезпечують скоординовані дії на всіх рівнях управління. Це дозволяє знизити ризики та мінімізувати наслідки надзвичайних ситуацій, забезпечуючи таким чином безпеку та захист громадян.

4.2 Загальні вимоги безпеки з охорони праці для користувачів ПК

Безпека праці при використанні персональних компютерів (ПК) включає комплекс заходів, спрямованих на забезпечення здорових і безпечних умов роботи, а також запобігання професійним захворюванням та травмам. Організація робочого місця включає наступні основні вимоги щодо робочої поверхні, крісла та обладнання:

- робочий стіл повинен мати достатньо місця для розміщення ПК, документів та інших необхідних матеріалів;
- монітор повинен розташовуватись прямо перед користувачем на відстані 50-70 см від очей, з верхнім краєм екрану на рівні очей або трохи нижче;
- клавіатура і миша повинні розташовуватись на одному рівні з ліктями користувача, що сидить;
- робоче крісло повинно бути зручним, з регульованою висотою та спинкою;
- користувач повинен мати можливість сидіти з прямою спиною, підтримкою для попереку і ногами, що торкаються підлоги або підставки;
- розташування елементів робочого місця повинно дозволяти вільні рухи та переміщення під час роботи з ПК.

До вимог щодо освітлення та середовища на робочому місці прийнято відносити те, що освітлення має бути достатнім, але без блисків на екрані монітора, відповідно джерела світла повинні розташовуватись так, щоб не було

прямого відблиску на екрані. Температура в приміщенні повинна бути в межах 18-24°C, а вологість повітря повинна бути в межах 40-60%.

Окрім цього користувач повинен дотримуватися правильної постави: прямий торс, ноги під прямим кутом, передпліччя паралельно підлозі. Кожні 60 хвилин роботи з ПК потрібно робити короткі перерви по 5-10 хвилин для зменшення навантаження на очі та хребет. Рекомендується виконувати вправи для очей і розтягування мязів спини та рук. Для психологічного комфорту робоче місце повинно бути організоване таким чином, щоб зменшити рівень шуму та інших факторів, що відволікають

Щодо безпечного користування електроприладами — всі електроприлади повинні бути заземлені та перевірені на безпеку. Користувачі повинні уникати використання пошкоджених кабелів та розеток. Згідно із правилами експлуатації необхідно вимикати ПК та інші пристрої під час тривалої перерви або закінчення робочого дня, а також не допускати контакту електроприладів з водою. Для захисту від електромагнітного випромінювання монітори повинні відповідати сучасним стандартам безпеки, що мінімізують електромагнітне випромінювання, а робоче місце має бути обладнане екранами або фільтрами для зниження випромінювань.

Дотримання загальних вимог безпеки з охорони праці для користувачів ПК є важливим для забезпечення здорових та безпечних умов праці. Правильна організація робочого місця, ергономіка, дотримання режиму праці та відпочинку, а також технічна безпека обладнання сприяють зменшенню ризиків професійних захворювань та підвищенню продуктивності праці.

4.3 Заходи захисту обладнання від статичної електрики

Статична електрика є однією з поширених проблем, що може негативно впливати на електронне обладнання. Вона виникає в результаті накопичення електричного заряду на поверхні матеріалів і може призводити до пошкодження чутливих електронних компонентів, порушення роботи пристроїв і навіть їх

повного виходу з ладу. Тому важливо вживати заходів для захисту обладнання від статичної електрики.

«Статична електрика утворюється через тертя між двома матеріалами, що викликає перенесення електронів з одного матеріалу на інший. Це може відбуватися під час переміщення людей або обєктів, контактів різних поверхонь або при роботі з електронними компонентами»[5].

Статичні розряди можуть спричиняти:

- Пошкодження електронних компонентів: Напруга, що виникає при статичному розряді, може перевищувати допустимі межі для мікросхем та інших чутливих елементів.

- Збої в роботі пристроїв: Навіть незначні розряди можуть викликати короткочасні збої в роботі обладнання.

- Вихід з ладу: Повторювані статичні розряди можуть призвести до повного виходу з ладу обладнання.

Заходи захисту від статичної електрики

1. Використання заземлювальних проводів для підключення корпусів пристроїв до системи заземлення, що забезпечує відведення статичного заряду до землі.

2. Використання спеціальних антистатичних килимків, підлогових покриттів і настільних підкладок, які запобігають накопиченню заряду. Нанесення антистатичних розчинів на поверхні, що можуть накопичувати статичний заряд.

3. Використання антистатичного одягу та взуття, які зменшують ризик виникнення статичних розрядів під час роботи з обладнанням. Носіння антистатичних браслетів, які забезпечують відведення заряду з тіла користувача.

4. Підтримка оптимального рівня вологості у приміщенні (40-60%), що зменшує ризик утворення статичної електрики. Використання зволожувачів повітря для підтримання необхідного рівня вологості.

5. Використання іонізаторів повітря, які нейтралізують статичний заряд у повітрі. Встановлення антистатичних фільтрів на вентиляційних системах.

6. Зберігання чутливих електронних компонентів у спеціальних антистатичних пакетах або контейнерах. Використання антистатичних підкладок під час транспортування обладнання.

7. Проведення інструктажів та навчання працівників щодо правил поводження з обладнанням та заходів захисту від статичної електрики. Регулярний контроль за дотриманням вимог з охорони праці та техніки безпеки.

Захист обладнання від статичної електрики є важливою складовою збереження його працездатності та довговічності. Впровадження комплексних заходів, таких як заземлення, використання антистатичних матеріалів, контроль вологості та навчання персоналу, допоможе мінімізувати ризики, повязані зі статичними розрядами. Забезпечення належного рівня захисту сприятиме стабільній та безперебійній роботі електронного обладнання, що є критично важливим для багатьох підприємств.

ВИСНОВКИ

У цій бакалаврській роботі проведено детальний аналіз та оцінку ефективності сучасних асиметричних криптографічних алгоритмів. Основну увагу приділено розгляду таких алгоритмів, як RSA, DSA та ECDSA, які широко використовуються для забезпечення безпеки та автентичності інформації в цифрових системах.

Проведений аналіз показав, що алгоритм RSA, незважаючи на свою високу надійність та стійкість до криптографічних атак, має суттєві обмеження, пов'язані з великою довжиною ключів. Це обумовлює високі вимоги до обчислювальних ресурсів і може знижувати продуктивність системи. Однак, RSA залишається одним з найпопулярніших алгоритмів завдяки своїй простоті реалізації та широкій підтримці в різних програмних та апаратних середовищах.

Алгоритм DSA демонструє ефективність у випадках, де важлива швидкість підписання та перевірки даних. Він менш вимогливий до обчислювальних ресурсів порівняно з RSA, але також потребує значних довжин ключів для забезпечення високого рівня безпеки. DSA є хорошим вибором для застосувань, де критичними є швидкість і продуктивність.

ECDSA, що базується на еліптичних кривих, виявився найефективнішим серед розглянутих алгоритмів завдяки своїй здатності забезпечувати високий рівень безпеки при використанні коротших ключів. Це дозволяє значно зменшити обчислювальні витрати та підвищити продуктивність систем, що є особливо важливим для мобільних пристроїв та інших обмежених середовищ. Швидкість створення та перевірки підписів в ECDSA робить його привабливим для багатьох сучасних застосувань.

Висновок роботи полягає в тому, що вибір асиметричного криптографічного алгоритму залежить від специфічних вимог та умов застосування. RSA підходить для систем, де пріоритетом є максимальна надійність і стійкість до атак, DSA - для застосувань, де критичними є швидкість і ефективність обчислень, а ECDSA - для середовищ з обмеженими ресурсами, які вимагають високої продуктивності та безпеки. Кожен з алгоритмів має свої

унікальні переваги та недоліки, і їх ефективність варіюється залежно від конкретних потреб та контексту використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про інформацію. Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/2657-12#Text (дата звернення: 01.03.2024).
2. Про захист інформації в інформаційно-комунікаційних системах. Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/80/94-вр#Text (дата звернення: 01.03.2024).
3. ДСТУ ISO/IEC 9798-3:2021 Інформаційні технології. Методи безпеки IT. Автентифікація об’єктів. Частина 3. Механізми з використанням методу цифрового підпису (ISO/IEC 9798-3:2019, IDT). БУДСТАНДАРТ Online - нормативні документи будівельної галузі України. URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=96872 (дата звернення: 06.03.2024).
4. Про захист населення і територій від надзвичайних ситуацій техногенного та природного характеру. Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/1809-14#Text (дата звернення: 08.05.2024).
5. Захист населення і територій від надзвичайних ситуацій – WikiLegalAid. Платформа правових консультацій - WikiLegalAid. URL: https://wiki.legalaid.gov.ua/index.php/Захист_населення_і_територій_від_надзвичайних_ситуацій (дата звернення: 08.05.2024).
6. Желібо Є. П., Заверуха Н. М., Зацарний В. В. Безпека життєдіяльності : підручник / ред. Є. п. Желібо. 6-те вид. Київ : Каравела, 2008. 338 с.
7. RSA-Based Signature Algorithms. Information Security Stack Exchange. URL: https://security.stackexchange.com/questions/59288/rsa-based-signature-algorithms (дата звернення: 06.03.2024).
8. National Institute of Standards and Technology. NIST. URL: https://www.nist.gov (дата звернення: 07.03.2024).

9. Karnaukhov, A., Tymoshchuk, V., Orlovska, A., & Tymoshchuk, D. (2024). USE OF AUTHENTICATED AES-GCM ENCRYPTION IN VPN. Матеріали конференцій МЦНД, (14.06. 2024; Суми Україна), 191-193. https://doi.org/10.62731/mcnd-14.06.2024.004Ез
10. fficient and Secure ECDSA Algorithm and its Applications: A Survey. arXiv.org. URL: https://arxiv.org/abs/1902.10313 (date of access: 10.03.2024).
11. Understanding the RSA algorithm. arXiv.org. URL: https://arxiv.org/abs/2308.02785 (дата звернення: 10.03.2024).
12. An Introduction to Digital Signature Schemes. arXiv.org. URL: https://arxiv.org/abs/1404.2820 (дата звернення: 19.06.2024).
13. Тимощук, В., & Стебельський, М. (2023). Шифрування даних в операційних системах. Матеріали VI Міжнародної студентської науково-технічної конференції „Природничі та гуманітарні науки. Актуальні питання “, 183-184.
14. Kuznetsov, A., Karpinski, M., Ziubina, R., Kandiy, S., Frontoni E., Veselska, O., & Kozak, R. (2023). Generation of nonlinear substitutions by simulated annealing algorithm. Information, 14(5), art. no. 259, 1-16. doi: 10.3390/info14050259.
15. Kovalchuk, O., Karpinski, M., Banakh, S., Kasianchuk, M., Shevchuk, R., & Zagorodna, N. (2023). Prediction machine learning models on propensity convicts to criminal recidivism. Information, 14(3), art. no. 161, 1-15. doi: 10.3390/info14030161.
16. Yesin, V., Karpinski, M., Yesina, M., Vilihura, V., Kozak, R., Shevchuk, R. (2023). Technique for Searching Data in a Cryptographically Protected SQL Database. Applied Sciences, 13(20), art. no. 11525, 1-21. doi: 10.3390/app132011525.

ДОДАТКИ

Додаток А Програмна реалізація цифрового підпису RSA

```
import random
from sympy import mod_inverse
from Crypto.Util import number

def generate_keypair(length):
  p = number.getPrime(length)
  q = number.getPrime(length)
  n = p * q
  phi = (p - 1) * (q - 1)

  # Choose an integer e such that e and phi(n) are coprime
  e = random.randrange(2, phi)
  g = gcd(e, phi)
  while g != 1:
    e = random.randrange(2, phi)
    g = gcd(e, phi)

  # Calculate d
  d = mod_inverse(e, phi)

  # Public key (e, n) and private key (d, n)
  return ((e, n), (d, n))

def gcd(a, b):
  while b != 0:
    a, b = b, a % b
  return a

def sign_message(private_key, message):
  d, n = private_key
  # Convert the message to a number based on the character and
sign it using a^d mod n
  signature = [pow(ord(char), d, n) for char in message]
  return signature

def verify_signature(public_key, message, signature):
  e, n = public_key
  # Verify the signature by converting it back to the original
message
  decrypted_message = ''.join([chr(pow(char, e, n)) for char in
signature])
  return decrypted_message == message

def main_rsa():
  length = 2048 # key length
  public_key, private_key = generate_keypair(length)

  message = "This is a secret message."
```

```
signature = sign_message(private_key, message)
verification = verify_signature(public_key, message, signature)
return verification
```

Додаток Б Програмна реалізація цифрового підпису DSA

```
from Crypto.Util import number
import hashlib
import random


def sign(message, p, q, g, x):
  # Generate a random k
  k = random.randint(1, q-1)

  # Calculate r
  r = pow(g, k, p) % q

  # Calculate hash of the message
  hash_obj = hashlib.sha1()
  hash_obj.update(message.encode())
  hash_value = int.from_bytes(hash_obj.digest(), 'big')

  # Calculate s
  k_inv = pow(k, -1, q)
  sing = (k_inv * (hash_value + x * r)) % q

  return r, sing

def verify(message, r, s, p, q, g, y):
  if not (0 < r < q) or not (0 < s < q):
    return False

  # Calculate hash of the message
  hash_obj = hashlib.sha1()
  hash_obj.update(message.encode())
  hash_value = int.from_bytes(hash_obj.digest(), 'big')

  # Calculate w
  w = pow(s, -1, q)

  # Calculate u1 and u2
  u1 = (hash_value * w) % q
  u2 = (r * w) % q

  # Calculate v
  v = ((pow(g, u1, p) * pow(y, u2, p)) % p) % q

  return v == r
def main_dsa():
  # Generate prime p and q (160 bits) such that p-1 is a multiple
of q
  while True:
    p = number.getPrime(512)
    q = number.getPrime(8)
    if (p - 1) % q == 0:
```

```
    break
# Generate g such that 1 < g < p
h = random.randint(2, p-2)
g = pow(h, (p-1)//q, p)
# Generate private key
x = random.randint(1, q-1)
# Generate public key
y = pow(g, x, p)
message = "This is a secret message."
# Signing the message
r, s = sign(message, p, q, g, x)
# Verifying the signature
is_valid = verify(message, r, s, p, q, g, y)
return is_valid
```

Додаток В Програмна реалізація цифрового підпису ECDSA

```
import hashlib
import random
from sympy import mod_inverse


# Elliptic curve parameters (secp256k1)
p =
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F
a = 0
b = 7
Gx =
0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798
Gy =
0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8
n =
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141

# Utility functions
def is_on_curve(point):
  """Check if a point is on the elliptic curve."""
  if point is None:
    return True
  x, y = point
  return (y ** 2 - x ** 3 - a * x - b) % p == 0


def point_add(point1, point2):
  """Add two points on the elliptic curve."""
  if point1 is None:
    return point2
  if point2 is None:
    return point1

  x1, y1 = point1
  x2, y2 = point2

  if x1 == x2 and y1 != y2:
    return None

  if x1 == x2:
    m = (3 * x1 ** 2 + a) * mod_inverse(2 * y1, p)
  else:
    m = (y2 - y1) * mod_inverse(x2 - x1, p)

  x3 = m ** 2 - x1 - x2
  y3 = y1 + m * (x3 - x1)

  return (x3 % p, -y3 % p)

def scalar_mult(k, point):
```

```
    """Multiply a point by an integer."""
    if k % n == 0 or point is None:
        return None
    if k < 0:
        return scalar_mult(-k, (-point[0], -point[1]))
    result = None
    addend = point
    while k:
        if k & 1:
            result = point_add(result, addend)
        addend = point_add(addend, addend)
        k >>= 1
    return result

# Key generation
def generate_keypair():
    d = random.randrange(1, n)
    Q = scalar_mult(d, (Gx, Gy))
    assert is_on_curve(Q), "Public key is not on the curve."
    return d, Q

# Signing
def sign_message(private_key, message):
    z = int.from_bytes(hashlib.sha256(message).digest(), 'big')
    r, s = 0, 0
    while r == 0 or s == 0:
        k = random.randrange(1, n)
        x, y = scalar_mult(k, (Gx, Gy))
        r = x % n
        s = (mod_inverse(k, n) * (z + r * private_key)) % n
    return r, s

# Verifying
def verify_signature(public_key, message, signature):
    r, s = signature
    if not (1 <= r < n and 1 <= s < n):
        return False
    z = int.from_bytes(hashlib.sha256(message).digest(), 'big')
    w = mod_inverse(s, n)
    u1 = (z * w) % n
    u2 = (r * w) % n
    x, y = point_add(scalar_mult(u1, (Gx, Gy)), scalar_mult(u2,
public_key))
    if not is_on_curve((x, y)):
        print("Computed point is not on the curve.")
    return (r % n) == (x % n)

def main_ecdsa():
    private_key, public_key = generate_keypair()
    message = b"This is a secret message."
    signature = sign_message(private_key, message)

    valid = verify_signature(public_key, message, signature)
    return valid
```

Додаток Г Програмна реалізація модуля measure.py для заміру ефективності алгоритмів

```
import psutil
import os
import timeit


def measure_performance(func, func_name, *args):
  process = psutil.Process(os.getpid())

  start_memory = process.memory_info().rss
  start_cpu_percent = psutil.cpu_percent(interval=None)

  execution_time = timeit.timeit(lambda: func(*args), number=1)

  end_memory = process.memory_info().rss
  end_cpu_percent = psutil.cpu_percent(interval=None)

  avg_cpu_percent = (start_cpu_percent + end_cpu_percent) / 2
  memory_used = end_memory - start_memory
  performance_metrics = {
    "function_name": func_name,
    'execution_time': execution_time,
    'current_memory_usage': memory_used,
    'cpu_usage': avg_cpu_percent,
  }
  return performance_metrics
```

Додаток Д Програмна реалізація модуля main.py

```
from rsa import main_rsa
from ecdsa import main_ecdsa
from dsa import main_dsa
from measure import measure_performance

print(measure_performance(main_rsa, "rsa"))
print(measure_performance(main_ecdsa, "ecdsa"))
print(measure_performance(main_dsa, "dsa"))
```