ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавр

на тему: «**Дослідження методів реалізації криптографічного алгоритму RSA (Rivest-Shamir-Adleman) з використанням бібліотеки OpenSSL**»

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
ступеня вищої освіти бакалавр
групи 126ІСТ_бд_2021
Розумій Антон Валентинович
Керівник: Одарущенко Олег
Миколайович
Рецензент: Муравльов Володимир
Вячеславович

Полтава – 2025 року

ВСТУП

У сучасному цифровому світі захист інформації є однією з найважливіших складових безпеки комунікацій, фінансових операцій, державних та корпоративних даних. Криптографія, як наука про шифрування, відіграє ключову роль у забезпеченні конфіденційності, цілісності та автентичності інформації. Серед різноманітних криптографічних алгоритмів особливе місце займає RSA (Rivest–Shamir–Adleman) – алгоритм з відкритим ключем, який став основою для багатьох сучасних технологій захисту даних.

Цифровий світ стоїть перед безпрецедентними викликами в галузі інформаційної безпеки. Серед фундаментальних технологій, що забезпечують захист даних, RSA займає центральне місце як механізм, що поклав початок сучасним системам шифрування з відкритим ключем. Його значення для кібербезпеки важко переоцінити – саме RSA лежить в основі безпечних електронних комунікацій, цифрових підписів та захисту конфіденційної інформації в мережах.

Актуальність дослідження алгоритму RSA зростає у міру збільшення складності кіберзагроз і попиту на кваліфікованих фахівців з інформаційної безпеки. У сучасних умовах, коли цифрові технології охоплюють усі сфери життя – від банківської справи до державного управління, – розуміння принципів роботи алгоритмів шифрування, зокрема RSA, стає важливою складовою професійної підготовки IT-спеціалістів.

Мета роботи – дослідження методів реалізації криптографічного алгоритму RSA з використанням бібліотеки OpenSSL. Особлива увага приділена поетапному вивченню процесів генерації ключів, шифрування, підписування повідомлень, перевірки цифрового підпису, а також аналізу сертифікатів формату X.509.

Об'єкт дослідження кваліфікаційної роботи – процеси розроблення криптографічних алгоритмів і протоколів.

Предмет дослідження – методичні підходи та інструменти вивчення криптографічного алгоритму RSA, включаючи його математичні основи, реалізацію та практичне застосування в інформаційних системах.

Методи дослідження: У процесі дослідження методів реалізації алгоритму RSA було застосовано комплексний підхід, що поєднує як теоретичний аналіз, так і практичну реалізацію. Теоретична частина ґрунтувалася на методах системного аналізу, які дозволили структуровано вивчити математичні основи RSA, його місце в сучасних криптографічних системах, а також вимоги до безпеки. Практична реалізація базувалася на методах розроблення програмно-алгоритмічного забезпечення із використанням мови програмування C та бібліотеки OpenSSL (через API і CLI). Для перетворення форматів даних і допоміжних операцій залучено мову Python. Методологія передбачала покрокову реалізацію основних криптографічних процедур: генерації ключів, шифрування, дешифрування, підписування повідомлень, перевірки цифрових підписів та сертифікатів формату X.509. Коректність результатів дослідження підтверджувалася шляхом перехресної перевірки, що забезпечує їх достовірність та відтворюваність.

Інформаційна база, що використовувалася: публікації в періодичних виданнях, вебресурси.

Практична значущість: полягає у створенні доступного і структурованого прикладу реалізації криптографічних процедур на основі алгоритму RSA, що може бути використано у навчальному процесі, підготовці курсів, лабораторних робіт та для особистого практичного досвіду студентів, адміністраторів і розробників.

Робота складається зі вступу, трьох розділів, висновків та списку літератури. Обсяг роботи становить 41 сторінку, 3 таблиці, 27 рисунків, 2 додатки.

РОЗДІЛ 1
АНАЛІЗ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ З ВІДКРИТИМ КЛЮЧЕМ. АЛГОРИТМ RSA (RIVEST-SHAMIR-ADLEMAN)

1.1 Теоретичні основи криптографії з відкритим ключем

Криптографія з відкритим ключем, або асиметрична криптографія, стала справжньою революцією у сфері безпечної комунікації, запропонувавши рішення для проблем, які довгий час були нерозв'язними в рамках традиційних симетричних систем. Її поява кардинально змінила підхід до конфіденційності, цілісності та автентифікації даних у цифровому світі. На відміну від симетричної криптографії, де для шифрування та дешифрування використовується один і той самий секретний ключ, асиметричні системи оперують парою математично пов'язаних ключів: відкритим (публічним) ключем та закритим (приватним) ключем. Ця фундаментальна відмінність відкриває нові можливості, які були недоступні для симетричних алгоритмів [1].

Основний принцип роботи асиметричної криптографії полягає в тому, що відкритий ключ може бути вільно розповсюджений і доступний будь-кому, хто бажає надіслати зашифроване повідомлення або перевірити цифровий підпис. Закритий же ключ, навпаки, повинен зберігатися в суворій таємниці його власником. Математичний зв'язок між цими ключами є таким, що зашифроване відкритим ключем повідомлення може бути розшифроване лише відповідним закритим ключем, і навпаки. Ця асиметрія дозволяє розв'язати проблему розподілу ключів, яка є однією з найскладніших у симетричній криптографії. Для обміну інформацією за допомогою симетричного шифрування відправнику та отримувачу необхідно заздалегідь узгодити спільний секретний ключ, що часто вимагає захищеного фізичного каналу або використання складних протоколів обміну ключами. Асиметрична криптографія усуває цю перешкоду, дозволяючи двом сторонам

встановлювати захищений зв'язок без попередньої зустрічі чи обміну секретами, використовуючи лише публічно доступні ключі.

Історично криптографія розвивалася переважно як симетрична. Протягом століть методи шифрування, від простих перестановок до складних машинних шифрів, базувалися на ідеї спільного секрету. Проте зі зростанням обсягів електронних комунікацій та необхідності взаємодії між великою кількістю невідомих один одному сторін, проблема безпечного обміну ключами ставала все більш актуальною.

Можливо уявити ситуацію, де мільйони користувачів Інтернету повинні безпечно спілкуватися з тисячами веб-сайтів. Кожен користувач не може мати унікальний секретний ключ з кожним сайтом, і тим більше неможливо фізично обмінятися цими ключами. Ця потреба у масштабованому та безпечному обміні ключами стала каталізатором для розробки криптографії з відкритим ключем у 1970 рр. Піонерами в цій галузі стали Вітфілд Діффі та Мартін Хеллман, які у 1976 р. представили концепцію асиметричних ключів та протокол обміну ключами, що нині носить їхні імена. Невдовзі після цього, у 1977 р., Рональд Рівест, Аді Шамір та Леонард Адлеман розробили алгоритм RSA, який став першим практичним та широко застосовуваним алгоритмом з відкритим ключем, здатним як до шифрування, так і до цифрового підпису [2].

Сфери застосування криптографії з відкритим ключем є надзвичайно широкими та охоплюють майже всі аспекти сучасного цифрового життя. Однією з ключових областей є електронний підпис. Завдяки асиметричній криптографії, відправник може «підписати» документ за допомогою свого закритого ключа, а будь-хто інший може перевірити справжність цього підпису, використовуючи відкритий ключ відправника. Це гарантує цілісність даних, підтверджуючи, що документ не був змінений після підпису, та автентифікацію відправника, доводячи, що саме він створив цей підпис. Електронні підписи є основою для юридично значущих електронних документів, безпечного банківського обслуговування та багатьох інших транзакцій.

Ще одна фундаментальна сфера – це обмін ключами. Хоча асиметричні алгоритми можуть безпосередньо шифрувати дані, вони, як правило, значно повільніші за симетричні алгоритми [3]. Тому на практиці асиметрична криптографія часто використовується для безпечної передачі симетричного ключа, який потім застосовується для шифрування основної маси даних. Це так звана гібридна криптографія. Відправник генерує випадковий симетричний ключ, шифрує його відкритим ключем отримувача, і передає разом із повідомленням, зашифрованим цим симетричним ключем. Отримувач дешифрує симетричний ключ своїм закритим ключем, а потім використовує його для дешифрування повідомлення. Цей підхід поєднує безпеку асиметричної криптографії для обміну ключами та ефективність симетричної криптографії для шифрування даних [4].

Безпосереднє шифрування даних за допомогою відкритого ключа є ще однією важливою функцією. Це дозволяє будь-кому надіслати конфіденційне повідомлення, яке може прочитати лише власник відповідного закритого ключа. Це широко застосовується в електронній пошті (наприклад, через протокол PGP/GPG), для захисту веб-трафіку (в протоколах TLS/SSL, що лежать в основі HTTPS), а також у багатьох інших системах, де необхідно забезпечити конфіденційність інформації, що передається через незахищені канали. Таким чином, криптографія з відкритим ключем не просто доповнила, а трансформувала ландшафт кібербезпеки, зробивши можливим безпечний та масштабований обмін інформацією в глобальних мережах [6]. Вона є невід'ємним стовпом сучасного цифрового суспільства, забезпечуючи довіру та безпеку в епоху повсюдної взаємодії.

Криптографія з відкритим ключем, або асиметрична криптографія, функціонує на принципово відмінних засадах порівняно із симетричною криптографією, що лежить в основі її здатності вирішувати проблеми масштабування та управління ключами. Фундаментальна відмінність полягає в тому, що симетричні системи використовують один і той самий секретний ключ як для операції шифрування, так і для операції дешифрування. Це

означає, що якщо Аліса хоче надіслати зашифроване повідомлення Бобу, і вони використовують симетричний алгоритм, їм обом необхідно володіти абсолютно однаковим секретним ключем. Цей ключ має бути відомий лише їм двом і нікому іншому, оскільки будь-яка третя сторона, яка отримає доступ до цього ключа, зможе безперешкодно дешифрувати їхні повідомлення. Такий підхід вимагає наявності захищеного каналу для первісного обміну ключем, що є значним логістичним викликом, особливо у великих та розподілених мережах. Наприклад, якщо Аліса та Боб перебувають на різних континентах і ніколи не зустрічалися, безпечний обмін ключем стає проблемою кур'єра або іншого довіреного, але потенційно вразливого методу.

На противагу цьому, асиметрична криптографія заснована на концепції пари ключів: відкритого (публічного) ключа та закритого (приватного) ключа. Ці ключі математично пов'язані, але володіють унікальними функціональними властивостями. Відкритий ключ, як випливає з його назви, призначений для публічного розповсюдження. Будь-хто може його отримати і використовувати. Закритий ключ, навпаки, є суто особистим і повинен зберігатися в таємниці його власником [7].

Ключовим принципом є те, що інформація, зашифрована відкритим ключем, може бути розшифрована лише відповідним закритим ключем. І навпаки, дані, підписані закритим ключем, можуть бути перевірені лише відповідним відкритим ключем. Ця властивість базується на так званих односторонніх функціях з «лазівкою». Це означає, що існує функція, яку легко обчислити в одному напрямку, але надзвичайно складно (обчислювально нездійсненно) обчислити у зворотному напрямку без знання додаткової секретної інформації – «лазівки». У контексті RSA такою функцією є множення двох великих простих чисел, що легко обчислити, але факторизація отриманого результату (пошук цих двох простих чисел) є дуже складною задачею без додаткових відомостей [8]. Закритий ключ фактично є цією «лазівкою», що дозволяє легко виконати зворотну операцію.

Ця фундаментальна архітектура породжує кілька ключових відмінностей між двома типами криптографії. По-перперше, у симетричних системах, як вже згадувалося, проблема розподілу ключів є центральною. Для кожного нового зв'язку або групи осіб, що бажають безпечно спілкуватися, потрібен окремий секретний ключ, і його необхідно безпечно передати всім учасникам. У великих мережах це призводить до експоненційного зростання кількості ключів, що необхідно управляти (для N учасників потрібно $N\times(N-1)/2$ унікальних ключів), що робить систему практично некерованою. Наприклад, якщо 1000 користувачів хочуть спілкуватися між собою, їм знадобиться майже півмільйона унікальних ключів.

Асиметрична криптографія радикально вирішує цю проблему. Кожен користувач генерує свою пару ключів (один відкритий, один закритий). Відкритий ключ публікується, а закритий зберігається в таємниці. Якщо Аліса хоче надіслати конфіденційне повідомлення Бобу, їй достатньо отримати його відкритий ключ. Їй не потрібно попередньо обмінюватися секретами з Бобом. Це значно спрощує управління ключами та робить можливим безпечне спілкування у великих, відкритих системах, таких як Інтернет.

Асиметрична криптографія дозволяє реалізувати функціонал, недоступний для симетричних систем, а саме незаперечний електронний підпис. У симетричній криптографії, якщо Аліса шифрує повідомлення ключем K і надсилає його Бобу, Боб знає, що повідомлення зашифровано ключем K, але він не може довести третій стороні, що це повідомлення надіслала саме Аліса, оскільки Ключ K відомий також і йому. Симетричний підпис не має властивості незаперечності, оскільки підписант і верифікатор володіють однаковим ключем. У асиметричних системах, якщо Аліса підписує документ своїм закритим ключем, будь-хто може перевірити цей підпис за допомогою її відкритого ключа. Оскільки лише Аліса володіє своїм закритим ключем, це є незаперечним доказом того, що саме вона створила підпис. Це критично важливо для таких застосувань, як юридично значущі цифрові документи, банківські транзакції та автентифікація програмного забезпечення.

Асиметричні алгоритми є обчислювально значно дорожчими та повільнішими порівняно з симетричними алгоритмами [9]. Операції шифрування та дешифрування з відкритим ключем вимагають таких як факторизація великих чисел (як у RSA) або задача дискретного логарифму. Через це асиметрична криптографія рідко використовується для безпосереднього шифрування великих обсягів даних.

Щоб краще зрозуміти відмінності між симетричною та асиметричною криптографією, наведемо їх порівняльну характеристику в таблиці 1.1.

Таблиця 1.1 – Порівняння симетричної та асиметричної криптографії

Ознака / характеристика	Симетрична криптографія	Асиметрична криптографія
Кількість ключів	Один і той самий секретний ключ	Відкритий та закритий ключ
Швидкість	Дуже висока, ефективна для великих обсягів	Значно нижча, обчислювально інтенсивна
Функціональність	Конфіденційність даних	Цілісність, автентифікація
Приклади алгоритмів	AES, DES, 3DES, Blowfish	RSA, ECC (ECDSA, ECDH)

Таким чином, асиметрична криптографія, попри свою обчислювальну інтенсивність, стала невід'ємним фундаментом сучасної кібербезпеки, дозволивши розв'язати проблему масштабованого обміну ключами та забезпечити незаперечність, що є критично важливим для довіри в цифровому світі.

1.2 Місце RSA в сучасній криптографії та сфери застосування

Алгоритм RSA, завдяки своїй надійності та добре вивченій математичній основі, здобув надзвичайну популярність і став одним з найбільш поширених стандартів у сфері асиметричної криптографії. Його широке впровадження у різноманітні протоколи та архітектури безпеки

свідчить про довіру, яку йому надає світова спільнота. Зокрема, RSA є невіддільною частиною:

– SSL/TLS (Secure Sockets Layer/Transport Layer Security): ці протоколи, що забезпечують безпечне з'єднання в мережі Інтернет, використовують RSA для встановлення захищеного каналу зв'язку, RSA застосовується для автентифікації серверів (за допомогою цифрових сертифікатів) та для обміну симетричними ключами, які потім використовуються для шифрування основного трафіку, це дозволяє браузерам користувачів безпечно взаємодіяти з веб-сайтами (HTTPs);

– PGP (Pretty Good Privacy): один з найвідоміших інструментів для шифрування електронної пошти та файлів, PGP активно використовує RSA як для шифрування сесійних ключів, так і для створення та перевірки цифрових підписів [10];

– цифрові сертифікати (X.509): основа сучасної інфраструктури відкритих ключів (PKI) [11], RSA є найпоширенішим алгоритмом, що використовується для генерації ключів, якими підписуються цифрові сертифікати, ці сертифікати засвідчують автентичність веб-сайтів, програмного забезпечення та інших цифрових сутностей;

– VPN (Virtual Private Network): у багатьох VPN-протоколах RSA застосовується для автентифікації та обміну ключами, забезпечуючи конфіденційність та цілісність даних при передачі через незахищені мережі.

Така широка інтеграція в ключові стандарти та протоколи робить RSA фундаментальним компонентом сучасної інформаційної безпеки.

Спектр застосування RSA надзвичайно широкий і охоплює майже всі аспекти цифрової взаємодії, де потрібна конфіденційність, цілісність або автентифікація:

– захист веб-трафіку (HTTPs): як вже зазначалося, RSA лежить в основі безпечного з'єднання з веб-сайтами, коли ви відвідуєте сайт з HTTPs, ваш браузер використовує відкритий ключ RSA сервера для шифрування сесійного ключа, який потім застосовується для шифрування всього трафіку;

– шифрування електронної пошти: для забезпечення конфіденційності електронних листів RSA дозволяє відправнику зашифрувати повідомлення за допомогою відкритого ключа одержувача, і лише власник закритого ключа може його розшифрувати;

– цифрові підписи: RSA є одним з основних алгоритмів для створення та перевірки цифрових підписів, це дозволяє підтвердити, що документ або файл був створений певним відправником і не був змінений після підписання, це критично важливо для цілісності програмного забезпечення, фінансових транзакцій та юридичних документів;

– безпечне завантаження програмного забезпечення та оновлень, розробники часто підписують своє програмне забезпечення та оновлення за допомогою RSA-підписів, це гарантує користувачам, що завантажений файл є оригінальним і не містить шкідливого коду;

– безпека електронної комерції: усі транзакції, що вимагають високого рівня безпеки, такі як платежі кредитною карткою в Інтернеті, часто використовують RSA для захисту конфіденційної фінансової інформації.

Ці приклади демонструють універсальність RSA і його здатність задовольняти різноманітні потреби в сфері інформаційної безпеки.

Незважаючи на свою довговічність і широке використання, RSA не є статичним алгоритмом і постійно адаптується до нових викликів та технологічних змін:

– збільшення розміру ключів: з розвитком обчислювальних потужностей зростає ризик того, що зловмисники зможуть факторизувати все більші числа, це вимагає постійного збільшення рекомендованого розміру ключів RSA (наприклад, перехід від 1024-бітних до 2048-бітних і 4096-бітних ключів), у свою чергу, призводить до збільшення обчислювальних витрат [12];

– квантові обчислення: найбільшим довгостроковим викликом для RSA є потенційний розвиток практичних квантових комп'ютерів, Алгоритм Шора, що виконується на квантовому комп'ютері, може ефективно факторизувати великі числа, тим самим повністю руйнуючи безпеку RSA, це стимулює

активні дослідження в галузі постквантової криптографії (PQC), яка розробляє алгоритми, стійкі до атак квантових комп'ютерів [13];

– гібридні криптосистеми: для перехідного періоду та для підвищення загальної безпеки часто використовуються гібридні підходи, де RSA може використовуватися для обміну ключами, а симетричні алгоритми (наприклад, AES) - для шифрування даних, існують також гібридні схеми, що комбінують RSA з постквантовими алгоритмами;

– оптимізація та апаратна реалізація: постійно розробляються методи оптимізації виконання RSA, як програмні, так і апаратні (наприклад, спеціалізовані криптографічні чипи), для підвищення продуктивності та енергоефективності.

RSA залишається домінуючим алгоритмом у багатьох сферах, його майбутнє значною мірою залежить від адаптації до нових технологічних реалій та інтеграції з новими, квантово-стійкими криптографічними рішеннями.

1.3 Математичні основи алгоритму RSA

Алгоритм RSA базується на фундаментальних поняттях теорії чисел, що й забезпечує його криптографічну стійкість. Без розуміння цих математичних принципів неможливо повною мірою оцінити як надійність, так і витонченість цього методу шифрування. Його основна ідея полягає у використанні односторонніх функцій із «лазівкою» - обчислень, які легко виконати в одному напрямку, але майже неможливо обернути без спеціальної інформації.

Ключові математичні концепції, що лежать в основі RSA, включають:

– прості числа: цілі числа, більші за 1, які діляться лише на 1 і на себе, для RSA вибираються два великих, випадкових і різні простих числа p та q, їхній вибір є критичним для безпеки;

– модуль N: обчислюється як добуток двох простих чисел:

$$N = p \cdot q \tag{1.1}$$

де p – перше велике просте число.

q – друге велике просте число;

Це значення використовується як модуль для всіх криптографічних операцій:

– функція Ейлера ϕ(N) (тоцієнт Ейлера): для N=p·q обчислюється за формулою:

$$N = p \cdot q \; \phi(N) = (p-1)(q-1) \tag{1.2}$$

де p – перше велике просте число.

q – друге велике просте число;

Це значення є кількістю натуральних чисел, менших за N і взаємно простих з N. Воно є секретним і використовується для знаходження приватного ключа:

– експонента e (експонента шифрування): вибирається ціле число e таке, що 1<e<ϕ(N) і e взаємно просте з ϕ(N) (тобто їхній найбільший спільний дільник gcd(e,ϕ(N))=1), часто обирають невеликі прості числа, такі як 65537, для оптимізації швидкості шифрування [7];

– секретний експонент d (експонента дешифрування): обчислюється як модульне обернене до e за модулем ϕ(N), тобто:

$$\mathrm{d} \cdot \mathrm{e} \equiv 1 (\mathrm{mod} \phi(\mathrm{N})) \tag{1.3}$$

де d – секретний експонент.

e – відкритий експонент.

ϕ(N) – функція Ейлера;

Це d є секретним ключем і знаходиться за допомогою розширеного алгоритму Евкліда, дозволяє «скасувати» операцію шифрування:

– модульна арифметика: всі обчислення в RSA виконуються за модулем N, це означає, що результатом операції є залишок від ділення на N, наприклад, X(modN).

– теорема Ейлера (для шифрування/дешифрування): на основі цієї теореми, для будь-якого повідомлення m (представленого числом) і зашифрованого тексту C, виконується:

$$(me)d \equiv m(modN) \tag{1.4}$$

де m – вихідне повідомлення (числове представлення);

e – відкритий експонент;

d – секретний експонент;

N – модуль.

Це гарантує, що дешифрування успішно відновлює початкове повідомлення.

Алгоритм RSA складається з ключових етапів: генерація ключів, шифрування та дешифрування. Кожен з цих етапів має свою послідовність математичних операцій, які будуть детально візуалізовані у блок-схемах.

Вибір правильних параметрів для алгоритма RSA [8] є критично важливим для забезпечення його криптостійкості та продуктивності.

Найважливішим параметром є довжина ключа, яка безпосередньо впливає на складність факторизації модуля N та, відповідно, на безпеку системи.

На практиці для забезпечення належного рівня безпеки рекомендується використовувати ключі довжиною не менше 2048 біт. Для систем з підвищеними вимогами до безпеки (наприклад, у банківських чи державних структурах) використовують ключі 3072 або навіть 4096 біт.

Розглянемо етап генерації ключів. Він включає вибір простих чисел, обчислення модуля, функції Ейлера та публічного і приватного ключів (рис 1.1).

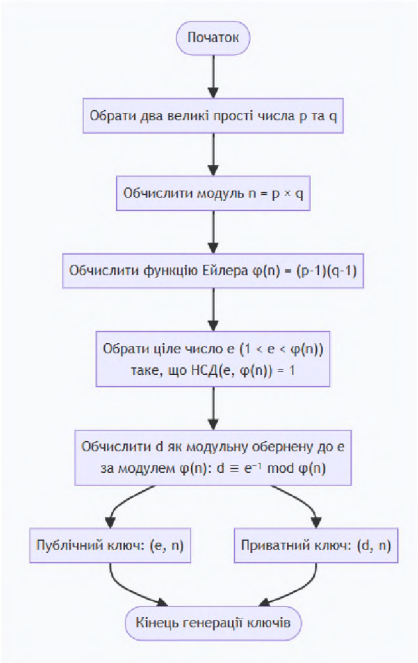

Рисунок 1.1 – Блок-схема етапу генерації ключів RSA

Після формування ключів, відбувається безпосереднє шифрування повідомлення. Для цього використовується публічний ключ і математична операція піднесення в ступінь за модулем (рис 1.2).

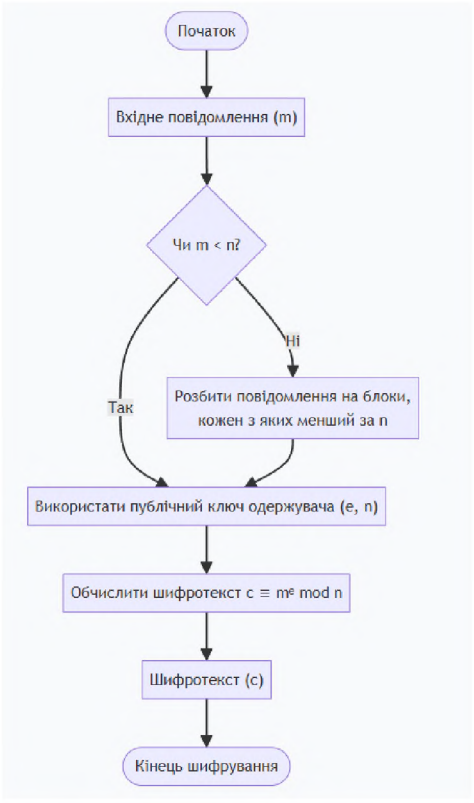

Рисунок 1.2 – Блок-схема етапу шифрування ключів RSA

Зашифроване повідомлення можна розшифрувати, застосовуючи приватний ключ. Процес дешифрування передбачає обчислення зворотної операції та відновлення оригінального тексту (рис 1.3).

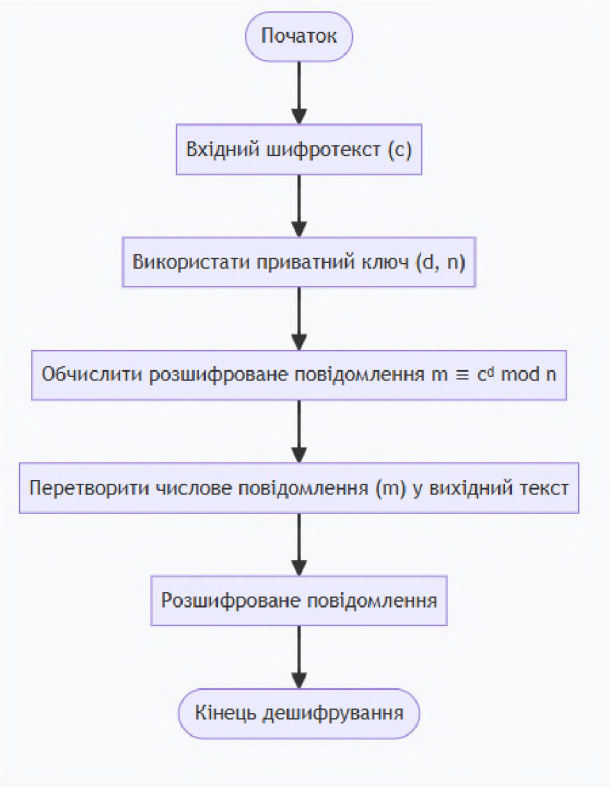

Рисунок 1.3 – Блок-схема етапу дешифрування повідомлення RSA

Алгоритм RSA залишається одним із наріжних каменів сучасної криптографії, забезпечуючи надійний захист конфіденційності та цілісності даних. Його стійкість, що ґрунтується на складності факторизації великих чисел, робить його незамінним для таких застосувань, як безпечна передача даних, електронний підпис та обмін ключами в корпоративних та глобальних мережах.

Детальний аналіз та візуалізація етапів генерації ключів, шифрування та дешифрування, представлені у цій роботі, підкреслюють елегантність та математичну глибину RSA. Розуміння цих процесів є критично важливим для розробки та впровадження ефективних криптографічних рішень, що відповідають сучасним вимогам інформаційної безпеки.

РОЗДІЛ 2
РОЗРОБЛЕННЯ ПОСЛІДОВНОСТІ ДОСЛІДЖЕННЯ АЛГОРИТМУ RSA. ІНСТРУМЕНТАРІЙ ДОСЛІДЖЕННЯ

2.1 Етапи дослідження алгоритму RSA

Для ефективного засвоєння теоретичних основ криптографічного алгоритму RSA важливо не лише ознайомитися з його математичними принципами, а й практично реалізувати ключові етапи: генерацію ключів, шифрування, дешифрування, створення та перевірку цифрового підпису, а також аналіз сертифікатів.

Саме тому наступним кроком у рамках кваліфікаційної роботи є розроблення чіткої послідовності дій для проведення дослідження алгоритму RSA, що включає побудову експериментального середовища, вибір інструментів і засобів, а також формування логічно обґрунтованого плану дій.

Важливим аспектом дослідження є визначення оптимального інструментарію, який дозволить реалізувати алгоритм у програмному середовищі, враховуючи як точність обчислень з великими числами, так і зручність для аналізу результатів.

У процесі підготовки експериментальної частини було вирішено орієнтуватися на мову програмування C у поєднанні з бібліотекою OpenSSL, яка забезпечує доступ до криптографічних функцій, зокрема до механізмів роботи з великими числами (Big Numbers), що критично важливо для реалізації RSA [13]. Також додатково використовуються утиліти Python для перетворення форматів даних (наприклад, ASCII ↔ HEX), а для роботи із сертифікатами застосовуються команди OpenSSL CLI.

У загальному вигляді процес дослідження RSA охоплює кілька ключових етапів, кожен з яких виконує певну функцію у формуванні цілісного

розуміння принципів роботи алгоритму. На рисунку 2.1 наведено узагальнену блок-схему, яка відображає логічну послідовність дій під час дослідження.

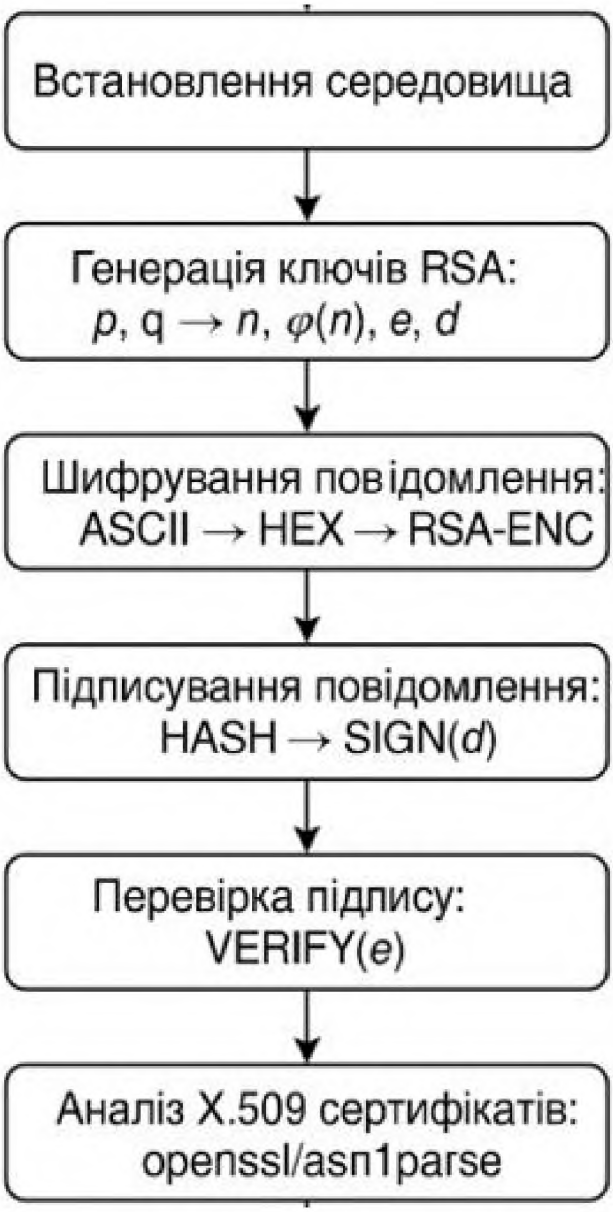

Рисунок 2.1 – Блок-схема послідовності дослідження алгоритму RSA

Схема демонструє логіку реалізації дослідження – від створення програмного середовища до аналізу сертифікатів, дозволяючи побачити загальну картину перед детальним розглядом кожного етапу.

Наступним кроком є деталізація інструментів, які були використані в ході реалізації. В таблиці 2.1 наведено короткий огляд програмного

забезпечення та технологій, що застосовувалися у рамках експериментальної частини дослідження RSA.

Таблиця 2.1 – Перелік інструментів, використаних для реалізації дослідження алгоритму RSA

Засіб	Призначення
OpenSSL (CLI та API)	Генерація ключів, шифрування, підписування, сертифікати
Мова програмування C	Реалізація логіки RSA та робота з великими числами
Python	Перетворення текстових даних (ASCII ↔ HEX)
Ubuntu Linux	Стабільне середовище виконання
asn1parse	Аналіз структури X.509 сертифікатів
OpenSSL Big Number API	Аритметика над великими числами

Завдяки такій поетапній структурі, дослідження RSA набуває не лише навчального, але й практичного значення, оскільки дозволяє моделювати реальні сценарії використання алгоритму в інформаційних системах. У наступному підрозділі буде докладно охарактеризовано використаний інструментарій, його особливості та переваги в контексті реалізації кожного з наведених етапів.

2.2 Деталізація етапів дослідження алгоритму RSA

Після визначення загальних етапів алгоритма RSA та необхідного інструментарію, доцільно розглянути кожен з етапів практичної реалізації алгоритму. Кожен етап має власну функцію в загальному процесі дослідження і реалізується за допомогою відповідного коду та математичних операцій.

Першим і базовим етапом є генерація ключової пари, яка складається з відкритого (публічного) та закритого (приватного) ключів. Це фундамент RSA, адже від правильності формування ключів залежить подальша безпека криптосистеми.

Основною задачею цього етапу є вибір двох великих простих чисел p і q, обчислення модуля $n = p \times q$, функції Ейлера $\varphi(n)$, вибір відкритої

експоненти e та знаходження закритої експоненти d, яка є оберненою до e за модулем φ(n) [14].

Після успішного формування ключів виконується етап шифрування повідомлення. У цьому процесі спочатку текст перетворюється в шістнадцяткове представлення (HEX), оскільки саме в такому вигляді з ним зручно працювати в математичних операціях. Потім цей HEX-код зашифровується за допомогою відкритого ключа (e, n).

На наступному етапі виконується дешифрування отриманого шифротексту. Для цього застосовується приватний ключ (d, n), після чого HEX-рядок перетворюється назад у початковий ASCII-текст. Таким чином, можна переконатися, що реалізація алгоритму працює коректно, якщо розшифроване повідомлення збігається з оригінальним.

Особливу увагу в дослідженні приділено реалізації цифрового підпису, адже це один із ключових механізмів забезпечення цілісності й автентичності даних. У цьому етапі повідомлення хешується, а потім створюється підпис шляхом піднесення хешу до ступеня d за модулем n. Перевірка підпису здійснюється обернено – хеш підписується відкритим ключем і порівнюється з оригіналом. Це дозволяє виявити навіть найменші зміни в повідомленні.

Окремим етапом є перевірка цифрових сертифікатів X.509, які використовуються в протоколах TLS/SSL. У рамках роботи було досліджено структуру таких сертифікатів, виділено ключові поля, а також реалізовано ручну перевірку підпису сертифіката за допомогою OpenSSL та власного коду на мові C.

Усі етапи показали коректність і працездатність реалізованих механізмів. Це підтверджує можливість використання власних криптографічних примітивів для демонстрації фундаментальних принципів безпеки даних.

У таблиці 2.2 представлено повну послідовність дій у рамках дослідження RSA, де кожен етап пов’язаний із конкретним результатом.

Таблиця 2.2 – Етапи реалізації дослідження RSA

Етап	Вхідні дані	Вихідні результати
Генерація ключів	Випадкові прості p,q	Пара ключів: відкритий (e, n), приватний (d, n)
Шифрування повідомлення	ASCII-текст, відкритий ключ	HEX-зашифроване повідомлення
Дешифрування	Зашифрований HEX	Відновлене повідомлення ASCII
Підписування повідомлення	Хеш повідомлення, приватний ключ	Цифровий підпис
Перевірка підпису	Підпис, відкритий ключ	Результат перевірки
Перевірка сертифіката	Сертифікат X.509	Статус перевірки підпису

Завдяки такій поетапній структурі, дослідження RSA набуває не лише навчального, але й практичного значення, оскільки дозволяє моделювати реальні сценарії використання алгоритму в інформаційних системах. У наступному підрозділі буде докладно охарактеризовано використаний інструментарій, його особливості та переваги в контексті реалізації кожного з наведених етапів.

2.3 Інструментарій дослідження

Ефективність та надійність реалізації криптографічного алгоритму безпосередньо залежить від вибору відповідного інструментарію. Особливо це актуально для RSA, оскільки цей алгоритм працює з надвеликими числами і потребує точних математичних операцій над ними. У цьому підрозділі розглянуто ті програмні засоби та технології, які були використані для реалізації дослідницької частини кваліфікаційної роботи, а також обґрунтовано доцільність їхнього вибору.

Насамперед слід зазначити, що дослідження реалізовувалося у середовищі операційної системи Linux (зокрема, дистрибутив Ubuntu), яка є зручною та широко підтримуваною платформою для розробки програмного забезпечення з відкритим кодом. Завдяки розвиненій екосистемі Linux значно спрощується встановлення необхідних бібліотек, таких як OpenSSL, і

забезпечується легкий доступ до криптографічних утиліт через командний рядок [15].

Окрему роль у дослідженні відіграє мова Python, яка використовувалась як допоміжний інструмент. Завдяки своїм вбудованим засобам обробки рядків і простим командам перетворення даних, Python став зручним для конвертації повідомлень у шістнадцятковий формат та зворотного декодування [16].

Python використовувався для генерації вхідних даних у зручному для обчислень форматі і що суттєво спростило підготовку тестових сценаріїв та автоматизацію перевірки результатів. Розробка та тестування функціоналу здійснювались у середовищі Linux, що забезпечує стабільність та доступ до необхідних бібліотек.

Ще одним важливим елементом інструментарію є утиліти OpenSSL CLI, які дозволяють здійснювати маніпуляції з сертифікатами X.509, включно з вилученням полів, перевіркою підписів, розбором структури ASN.1. Це було особливо корисно при реалізації етапу ручної перевірки цифрових сертифікатів.

Для наочності, наведено узагальнену таблицю 2.3, яка містить перелік інструментів, що використовувалися в ході дослідження, а також короткий опис їхнього функціонального призначення.

Таблиця 2.3 – Інструментарій реалізації дослідження RSA

Засіб / технологія	Призначення
Ubuntu Linux	Базове середовище виконання та компіляції
C (GCC компілятор)	Реалізація алгоритму на низькому рівні
OpenSSL BIGNUM API	Обчислення з великими числами
Python	Допоміжні обчислення та конвертація ASCII ↔ HEX
OpenSSL CLI (openssl x509)	Перевірка цифрових сертифікатів, розбір ASN.1-структури
asn1parse	Декодування тіла сертифікатів X.509 для ручного аналізу

Вибір саме цих інструментів дозволив реалізувати не лише теоретичну, а й прикладну частину дослідження. Застосування OpenSSL як на рівні API, так і через командний рядок, забезпечило гнучкість і відповідність сучасним

стандартам безпеки, а поєднання мов C і Python – оптимальну ефективність обчислень і зручність при підготовці даних [17].

Таким чином, сформований інструментарій повністю відповідає цілям дослідження й дозволяє повноцінно реалізувати кожен з етапів дослідницької частини, зберігаючи при цьому високий рівень достовірності та наочності результатів.

2.4 Перевірка достовірності та надійності результатів дослідження

У будь-якому криптографічному дослідженні важливо не лише реалізувати алгоритм, а й переконатися в коректності, достовірності та повторюваності отриманих результатів.

Зокрема, йдеться про те, щоб усі криптографічні операції (генерація ключів, шифрування, підпис, перевірка тощо) виконувалися згідно з математичними принципами, а результати були стабільними та відтворюваними незалежно від зовнішніх умов.

Одним із головних критеріїв достовірності є відповідність реалізації стандартам RSA [18]. Усі обчислення в рамках дослідження виконувалися відповідно до класичних математичних формул RSA, включно з використанням функції Ейлера, розширеного алгоритму Евкліда для пошуку оберненого елемента, та експоненціального шифрування у модульній арифметиці. Це гарантує, що поведінка алгоритму у дослідженні збігається з теоретичною моделлю.

Для підвищення впевненості у правильності реалізації було застосовано перехресну перевірку результатів. Наприклад:

– після виконання шифрування і дешифрування повідомлення результат порівнювався з оригіналом;

– хеш повідомлення перевірявся як вручну, так і автоматизованими командами OpenSSL;

– підпис повідомлення перевірявся як через власну програму, так і за допомогою OpenSSL CLI.

Також було враховано принцип повторюваності експериментів – усі кроки реалізовувалися у вигляді окремих програмних файлів (task1.c, task2.c тощо), які можуть бути повторно скомпільовані та виконані на будь-якому сумісному середовищі. Такий підхід забезпечує відтворюваність дослідження на інших комп'ютерах, незалежно від конкретного обладнання або конфігурації.

Ще одним важливим аспектом є аналіз граничних умов. У рамках дослідження, попри демонстраційний характер деяких прикладів (зокрема, використання коротших ключів), особливу увагу було приділено наголошенню на важливості дотримання безпечної довжини ключів у реальних системах (не менше 2048 біт). Це дозволяє критично осмислювати результати дослідження та розуміти обмеження спрощених моделей [23].

Таким чином, реалізована методика дослідження алгоритму RSA є не лише послідовною, але й достовірною, відтворюваною та адаптованою для майбутнього використання як у навчальних цілях, так і для прототипування реальних систем інформаційного захисту.

2.5 Процедура генерації ключів

Процедура генерації ключів в алгоритмі RSA є першим і фундаментальним кроком до встановлення безпечної комунікації. Вона передбачає створення пари математично пов'язаних ключів – відкритого (публічного) ключа та закритого (приватного) ключа.

Безпека RSA значною мірою залежить від належного виконання цієї процедури та вибору адекватної довжини ключів.

Процес генерації ключів складається з таких етапів:

– вони повинні бути випадково згенеровані, щоб ускладнити атаки на основі передбачення;

– вони повинні бути приблизно однієї довжини, але не надто близькими, щоб уникнути атак, заснованих на факторизації чисел, близьких до квадрата;

– генерації цих чисел використовують спеціальні алгоритми тестування на простоту, оскільки детермінована перевірка простоти для таких великих чисел є обчислювально нездійсненною.

Значення $\phi(n)$ є критично важливим для обчислення закритого ключа, але воно повинно зберігатися в таємниці, оскільки його розкриття дозволяє легко визначити p та q, тим самим руйнуючи безпеку системи.

Після всіх обчислень формується пара ключів:

– відкритий ключ (Public Key): складається з пари (e,n), цей ключ може бути вільно розповсюджений і є публічно доступним.

– закритий ключ (Private Key): складається з пари (d,n), цей ключ повинен зберігатися в суворій таємниці його власником, для закритого ключа також зберігаються p, q та $\phi(n)$ для оптимізації певних операцій, але основною його частиною є d.

На сьогоднішній день не існує ефективного (поліноміального за часом) алгоритму для факторизації дуже великих складених чисел на їх прості множники. Це означає, що для достатньо великих n (наприклад, 2048-бітних), навіть найпотужніші сучасні комп'ютери потребували б мільярди років, щоб перебрати всі можливі варіанти і знайти p та q [24].

Саме ця обчислювальна складність і є «лазівкою» RSA: множити прості числа легко, а ось розкладати великий добуток на множники надзвичайно складно. Таким чином, знання n не дозволяє зловмиснику легко знайти p та q, а отже, обчислити $\phi(n)$ і, як наслідок, визначити закритий ключ d. Це забезпечує криптографічну стійкість RSA, роблячи його надійним для захисту інформації в сучасному світі.

2.6 Практичне застосування та сучасний стан RSA

Практичне застосування RSA охоплює широкий спектр сфер, де потрібна конфіденційність, цілісність даних та автентифікація:

– RSA є основою для шифрування даних, що передаються по незахищених каналах, RSA не завжди використовується для прямого шифрування великих обсягів даних через його відносну повільність порівняно з симетричними алгоритмами (такими як AES), він є ключовим для обміну симетричними ключами [29],наприклад, у протоколах TLS/SSL (Transport Layer Security/Secure Sockets Layer), які забезпечують безпеку веб-трафіку (HTTPS), RSA використовується для шифрування симетричного ключа сесії, який потім застосовується для шифрування фактичних даних;

– RSA є частиною створення та верифікації цифрових підписів. Це дозволяє одержувачу повідомлення перевірити його автентичність та цілісність, тобто переконатися, що повідомлення дійсно надіслане заявленим відправником і не було змінене під час передачі, у практичних сценаріях, таких як підписання програмного забезпечення, електронних документів або електронних листів, RSA застосовується для підпису хеш-значення даних, а не самих даних, що значно підвищує ефективність [30];

– RSA відіграє центральну роль у структурі X.509 сертифікатів, які є фундаментальним компонентом інфраструктури відкритих ключів (PKI), X.509 пов'язують публічний ключ з певною ідентичністю (наприклад, веб-сайтом або особою) і підписуються довіреним центром сертифікації (CA) [26], це дозволяє користувачам перевіряти легітимність публічних ключів, гарантуючи, що вони спілкуються з правильним джерело, перевірка сертифікату X.509, як це демонструється в лабораторній роботі, включає вилучення публічного ключа емітента, підпису та тіла сертифіката, а потім використання цих даних для підтвердження дійсності підпису;

– безпека електронної пошти: RSA використовується в стандартах, таких як PGP (Pretty Good Privacy) та S/MIME (Secure/Multipurpose Internet Mail

Extensions), для шифрування електронних листів та накладання цифрових підписів, забезпечуючи конфіденційність та автентифікацію кореспонденції;

– у протоколах, таких як SSH (Secure Shell), RSA використовується для автентифікації клієнтів та серверів, а також для обміну ключами, що забезпечує безпечний канал для віддаленого управління системами.

Хоча RSA залишається критично важливим компонентом сучасної криптографії, його «сучасний стан» еволюціонує з огляду на зростаючі обчислювальні потужності та розвиток криптоаналізу. Безпека RSA безпосередньо залежить від довжини ключів. На сьогоднішній день, для достатнього рівня безпеки рекомендується використовувати ключі RSA довжиною 2048 біт або 3072 біти. Ключі меншого розміру, наприклад, 1024-бітні, вважаються недостатньо безпечними для довгострокового використання, оскільки їх потенційно можна зламати за допомогою сучасних обчислювальних ресурсів. У лабораторній роботі використовувалися менші ключі (128 біт) для простоти демонстрації, але підкреслюється, що для практики вони повинні бути набагато довшими. Безпека RSA базується на складності факторизації великих чисел. Доки не буде знайдено ефективного алгоритму факторизації (наприклад, за допомогою квантових комп'ютерів, що потенційно можуть запустити алгоритм Шора), RSA залишається стійким. Однак, постійно ведуться дослідження у сфері квантової криптографії (post-quantum cryptography), які можуть запропонувати альтернативні алгоритми, стійкі до атак квантових комп'ютерів [28].

Реалізація RSA алгоритму, особливо для обчислень з великими числами, вимагає використання спеціалізованих бібліотек, таких як OpenSSL Big Number library. Ці бібліотеки надають API для виконання арифметичних операцій (додавання, віднімання, множення, піднесення до степеня, модульні операції) над числами, які значно перевищують розміри стандартних цілих типів. Це дозволяє програмістам, використовуючи мови на кшталт C, працювати з числами довжиною 512 біт і більше, які є типовими для RSA.

У більшості сучасних застосувань RSA використовується як частина гібридної криптосистеми. Це означає, що RSA застосовується для обміну симетричними ключами (які є набагато ефективнішими для шифрування великих обсягів даних), а вже потім дані шифруються за допомогою симетричного алгоритму. Це поєднує переваги обох типів криптографії: безпечний обмін ключами RSA та високу швидкість симетричного шифрування.

Важливо відзначити, що, як і будь-який криптографічний алгоритм, RSA може мати вразливості, пов'язані з його неправильною реалізацією (наприклад, використання слабких генераторів випадкових чисел для створення простих чисел p і q), або з певними типами атак (наприклад, атаки по бічному каналу). Тому критично важливо використовувати перевірені, надійні бібліотеки (як OpenSSL) та дотримуватися найкращих практик безпеки [27].

Підсумовуючи, RSA залишається фундаментальним стовпом сучасної криптографії, що широко застосовується для шифрування, цифрових підписів та забезпечення безпеки комунікацій в Інтернеті через PKI. Його безпека залежить від довжини ключа та правильності реалізації, а майбутні виклики пов'язані з розвитком квантових обчислень спонукають до дослідження пост-квантових альтернатив.

РОЗДІЛ 3
РОЗРОБЛЕННЯ МЕТОДИЧНОГО ЗАБЕЗПЕЧЕННЯ ДОСЛІДЖЕННЯ АЛГОРИТМУ RSA

3.1 Практична реалізація та перевірка алгоритму RSA

Спочатку встановлюємо необхідні бібліотеки для роботи з великими числами (Big Numbers), як показано на рисунку 3.1

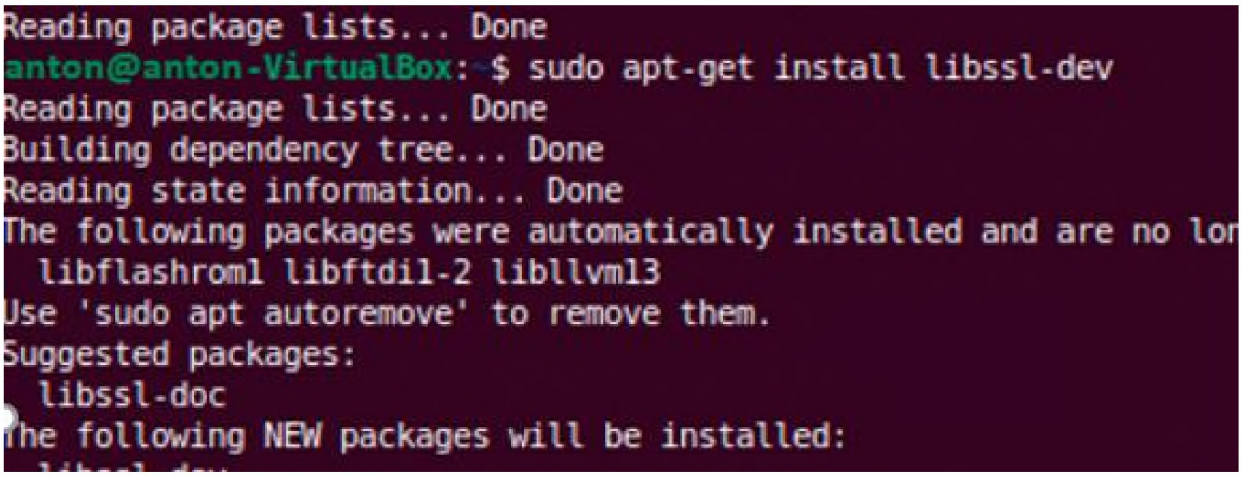

Рисунок 3.1 – Встановлення необхідних бібліотек для роботи з великими числами

Створюємо файл task1.c, в якому реалізовано обчислення приватного ключа d згідно з RSA-алгоритмом (рис. А.1, додаток А).

Після компіляції та запуску програми отримуємо значення приватного ключа (рис. 3.3).

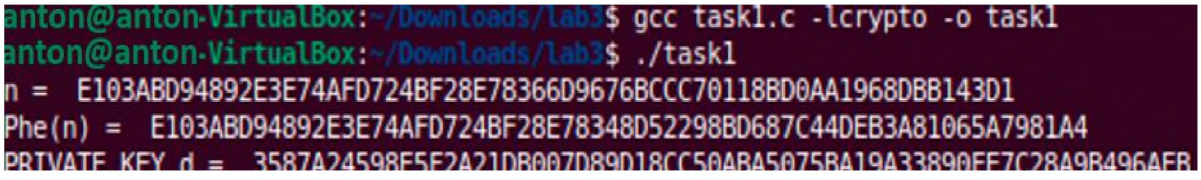

Рисунок 3.3 – Отримання значення приватного ключа після компіляції та запуску програми

Потрібно зашифрувати фразу «A top secret!». Для початку потрібно конвертувати цей ASCII-рядок в hex-рядок. Для цього можна використати наступну python команду (рис. 3.4).

```
$ python -c 'print("A top secret!".encode("hex"))'
4120746f702073656372657421
```

Рисунок 3.4 – Конвертація ASCII-рядка в hex-рядок

Файл task2.c (рис. А.2, додаток А). Для зашифровки і розшифровки тексту використовується функція BN_mod_exp().

Результат виконання коду (рис. 3.6).

```
anton@anton-VirtualBox:~/Downloads/lab3$ gcc task2.c -lcrypto -o task2
anton@anton-VirtualBox:~/Downloads/lab3$ ./task2
encryption of message =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75C
decryption of message =  4120746F702073656372657421
```

Рисунок 3.6 – Результат виконання коду шифрування

Для розшифрування створюєм файл task3.c (рис. А.3, додаток А), який також використовує BN_mod_exp() з приватним ключем.

Результат виконання коду (рис. 3.8).

```
anton@anton-VirtualBox:~/Downloads/lab3$ gcc task3.c -lcrypto -o task3
anton@anton-VirtualBox:~/Downloads/lab3$ ./task3
decryption of message =  50617373776F726420697320646565 73
```

Рисунок 3.8 – Результат виконання коду дешифрування

Після цього запускаємо наступну python команду, щоб перетворити hex-рядок назад в ASCII (рис. 3.9). Зашифрована фраза – «Password is dees»

```
anton@anton-VirtualBox:~/Downloads/lab3$ python3 -c 'print(bytes.fromhex("50617373776F726420697320646565 73")
.decode())'
Password is dees
```

Рисунок 3.9 – Перетворення hex-рядка назад в ASCII

Перетворюєм два рядки – «I owe you $2000.» та «I owe you $3000.» –у hex-формат (рис. 3.10). Хоча відрізняються вони лише одним байтом, після підпису отримуються абсолютно різні результати, що демонструє надійність механізму цифрового підпису.

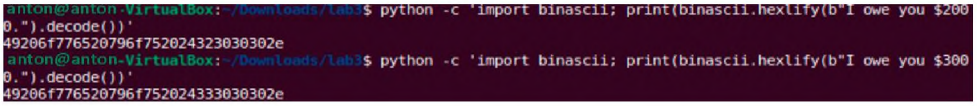

Рисунок 3.10 – Перетворення рядків в hex-формат для підписання

Створюєм файл task4.c для перевірки цифрового підпису (рис. A.4, додаток A).

Після запуску з обома варіантами повідомлення спостерігаєм, що навіть найменша зміна у тексті призводить до того, що підпис не проходить перевірку (рис. 3.12).

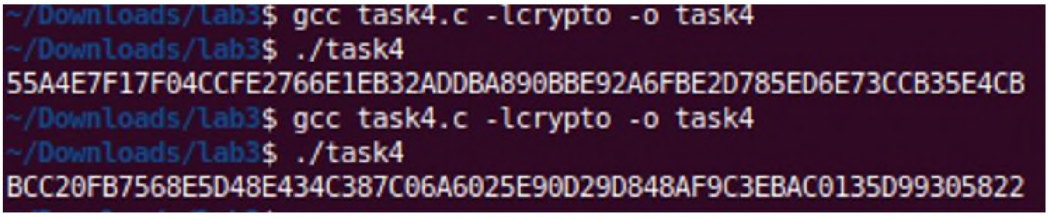

Рисунок 3.12 – Результат перевірки підпису при зміні тексту повідомлення

Створюєм файл task4.c (рис. A.5, додаток A).

Отримуєм hex-рядок (рис. 3.14).

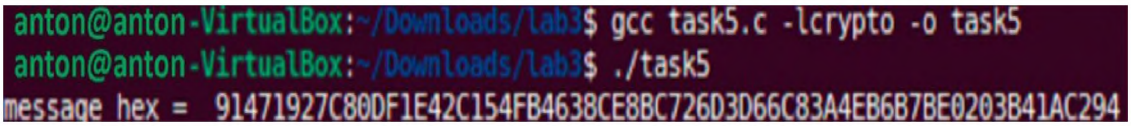

Рисунок 3.14 – Отримання hex-рядка

Додатково змінюєм один байт у підписі (наприклад, F2 на F3) і ще раз запускаю програму (рис. 3.15), щоб підтвердити, що підробка виявляється.

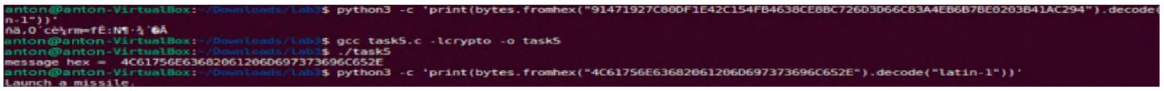

Рисунок 3.15 – Результат перевірки підпису при зміні одного байта в підписі.

Для початку завантажуєм сертифікати з сайту nytimes.com (рис. 3.16).

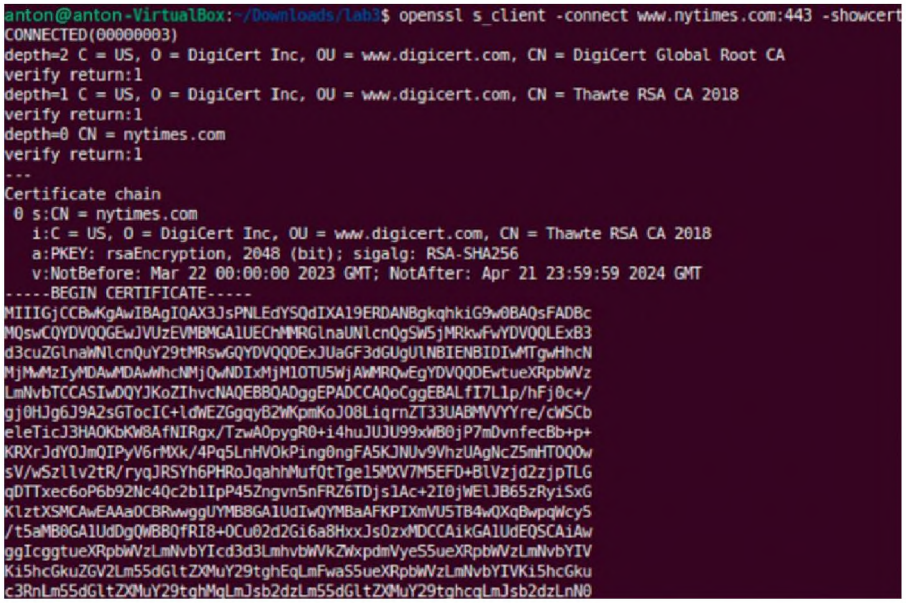

Рисунок 3.16 – Завантаження сертифікатів з сайту nytimes.com.

Зберігаєм їх в файли c0.pem та c1.pem (рис. 3.17).

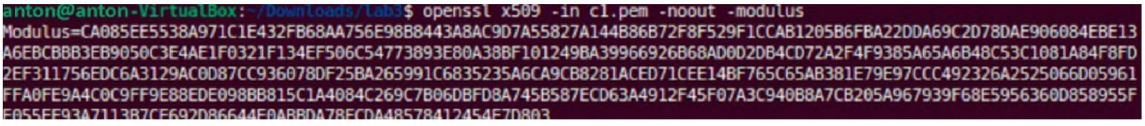

Рисунок 3.17 – Збереження сертифікатів у файли c0.pem та c1.pem

За допомогою команди openssl виділяю підпис і тіло сертифіката (рисунки 3.18–3.19).

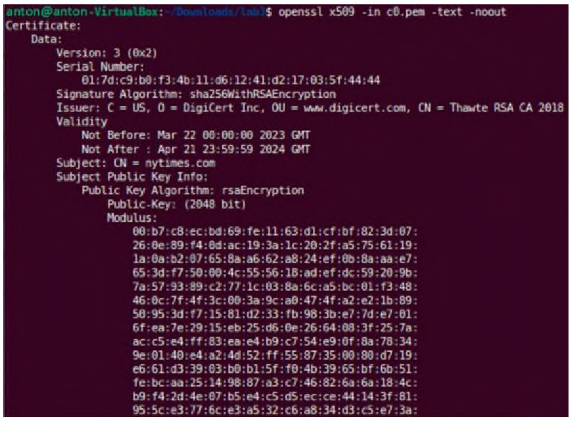

Рисунок 3.18 – Виділення підпису та тіла сертифіката за допомогою команди openssl (перша частина)

Рисунок 3.19 – Виділення підпису та тіла сертифіката за допомогою команди openssl (друга частина)

Після цього використаємо asn1parse для збереження в файлі c0_body.bin (рис. 3.20).

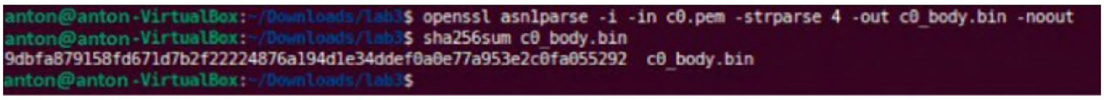

Рисунок 3.20 – Збереження в файлі c0_body.bin за допомогою asn1parse

Далі створюєм файл task6.c, в якому реалізована перевірка підпису сертифіката (рис. А.6, додаток А).

Незважаючи на невеликі відмінності у хеші, видно, що кінцева частина збігається (рис. 3.22), що вказує на правильність виконання перевірки.

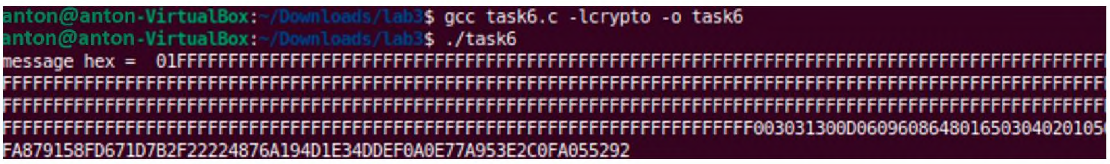

Рисунок 3.22 – Результат перевірки підпису сертифіката

Було розроблено методичне забезпечення для дослідження алгоритму RSA, що охоплює всі основні етапи його функціонування: генерацію ключів, шифрування, дешифрування, підписування повідомлень та перевірку цифрового підпису. Практична реалізація за допомогою мов програмування C та Python дозволила детально розглянути використання бібліотек для роботи з великими числами та застосування криптографічних функцій, зокрема BN_mod_exp().

Особливу увагу приділено демонстрації незаперечності цифрового підпису та аналізу справжності сертифікатів X.509. Отримані результати підтверджують коректність реалізації RSA та його придатність для забезпечення конфіденційності, автентичності та цілісності інформації у практичних системах інформаційної безпеки.

3.2 Техніко-економічне обгрунтування дослідження алгоритму RSA

У рамках даної кваліфікаційної роботи було проведено дослідження криптографічного алгоритму RSA з використанням бібліотеки OpenSSL. Для обґрунтування доцільності такого впровадження важливо розглянути його техніко-економічні аспекти – вартість розробки, затрати ресурсів, можливості масштабування, а також очікувані вигоди.

Для реалізації алгоритму RSA було використано інструменти з відкритим вихідним кодом: операційна система Ubuntu, бібліотека OpenSSL, мови програмування C та Python.

Це значно знижує витрати на ліцензії та дозволяє вільно адаптувати програмне забезпечення під конкретні потреби. Реалізація може виконуватись на стандартному апаратному забезпеченні без потреби в додаткових витратах на обладнання.

Відповідно до методичних рекомендацій, техніко-економічне обґрунтування передбачає:

– оцінку прямих витрат на реалізацію;

– аналіз непрямих витрат (енерговитрати, інфраструктура);

– визначення очікуваних вигод;

– співвідношення витрат та результатів;

– висновок щодо ефективності.

У таблиці наведено основні параметри, що використовувались у розрахунках:

Таблиця 3.1 – Вихідні дані для техніко-економічного аналізу

Показник	Значення
Погодинна ставка IT-спеціаліста-початківця	75 грн/год
Час, витрачений на реалізацію RSA	60 год
Середнє споживання енергії ноутбука	60 Вт
Вартість електроенергії	4,32 грн/кВт

Розрахунок витрат, загальні витрати на реалізацію оцінювались за формулою:

$$Сзаг = Спраці + Сел \quad (3.1)$$

де $Сзаг$ – загальні витрати на реалізацію дослідження, грн;

$Спраці$ – витрати на оплату праці виконавця, грн;

$Сел$ – витрати на спожиту електроенергію, грн.

Оплата праці:

$$Спраці = T \times Сгод = 60 \times 75 = 4500 \text{ грн} \quad (3.2)$$

де T – кількість годин, витрачених на виконання проєкту (60 год);

$Сгод$ – вартість однієї години роботи спеціаліста (75 грн/год).

Енергоспоживання:

$$Спраці = T \times Сгод = 60 \cdot 75 = 4500 \text{ грн} \quad (3.3)$$

де E – загальна спожита електроенергія, кВт·год;

P – потужність комп'ютера, кВт (0,06 кВт);

t – тривалість роботи, год (60 год).

$$Сел = E \cdot Сен = 3{,}6 \cdot 4{,}32 = 15{,}55 \text{ грн} \quad (3.4)$$

де $Сен$ – вартість 1 кВт·год електроенергії (2,64 грн/кВт·год);

$Сел$ – витрати на електроенергію, грн;

Підсумковий розрахунок:

$$Сзаг = 4500 + 15{,}55 = 4515{,}55 \text{ грн} \quad (3.5)$$

Розроблений програмний комплекс може бути:

– використаний як навчальний модуль у курсах криптографії, IT-безпеки;

– повторно застосований у лабораторних роботах;

– основою для подальших досліджень з шифрування.

Очікувана непряма вигода полягає у багаторазовому повторному використанні програмного модуля без потреби у доопрацюваннях. Таким чином, витрати можуть бути розподілені на десятки циклів використання, що значно підвищує питомий коефіцієнт ефективності.

Згідно з розрахунками та за умови повторного використання в освітньому процесі, розробка криптографічного програмного модуля є економічно доцільною. Витрати в обсязі 4515,55 грн повністю виправдані з огляду на отримані результати, гнучкість використання, масштабованість, а також високу освітню та методичну цінність проєкту.

ВИСНОВКИ

В рамках кваліфікаційної роботи було проведено комплексне дослідження та розробку, зосереджені на криптографічному алгоритмі RSA.

Після ретельного аналізу криптографічних алгоритмів з відкритим ключем, детального дослідження алгоритму RSA та розроблення послідовності й методичного забезпечення його дослідження, можна зробити наступні висновки:

RSA відіграє фундаментальну роль у сучасній криптографії з відкритим ключем, забезпечуючи конфіденційність, цілісність та автентифікацію даних у мережевих комунікаціях.

Було здійснено глибокий аналіз криптографічних алгоритмів з відкритим ключем, приділяючи особливу увагу алгоритму RSA (Rivest-Shamir-Adleman). Детально розглянуто принципи функціонування RSA, його математичні основи, а також ключові переваги та потенційні вразливості, що робить його одним із наріжних каменів сучасної асиметричної криптографії.

Було розроблено послідовность дослідження алгоритму RSA та визначено необхідний інструментарій. Було сформульовано чіткий план експериментів, який дозволить систематично вивчати різні аспекти продуктивності та безпеки RSA. Визначений конкретний перелік програмних засобів, криптографічних бібліотек, мов програмування та віртуальних машин, які будуть використовуватися для практичної реалізації дослідження, забезпечуючи відтворюваність та надійність отриманих результатів.

Розроблено методику дослідження алгоритму RSA. Методика забезпечення включає опис етапів дослідження, метрик вимірювання продуктивності, процедур збору та аналізу даних, а також очікуваних результатів. Такий підхід гарантує систематичність, ефективність та об'єктивність проведеного дослідження.

Загалом, проведений аналіз та розроблене методичне забезпечення надають комплексний підхід до вивчення та оцінки алгоритму RSA. Ця робота

не лише поглиблює розуміння функціонування одного з найважливіших криптографічних алгоритмів, але й створює практичну основу для подальших досліджень у сфері безпеки інформаційних систем.

Напрямком подальших досліджень є інтеграція алгоритму RSA з гібридними криптографічними системами, дослідження його стійкості до квантових атак, оптимізація обчислювальних процесів для пристроїв з обмеженими ресурсами (наприклад, веб-застосунки, мобільні платформи) а також поступовий перехід до постквантових криптографічних стандартів. Такий розвиток сприятиме підвищенню загального рівня захищеності сучасних інформаційно-комунікаційних систем.