Міністерство освіти і науки України

Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий
інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

здобувача Субач Олександр Андрійович
(ПІБ)

академічної групи 123-21-1
(шифр)

спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)

за освітньо-професійною програмою 123 Комп'ютерна інженерія
(офіційна назва)

на тему "Комп'ютерна підсистема перевірки цілісності та достовірності файлів при передачі даних"
(назва за наказом ректора)

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	доц.Каштан В.Ю.			
загального розділу	доц.Каштан В.Ю.			
спеціального розділу	доц.Каштан В.Ю.			
Рецензент				
Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)
_____Гнатушенко В.В.
(підпис) (прізвище, ініціали)

"25" січня 2025 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавр

здобувача Субач О.А. **академічної групи** 123-21-1
(прізвище та ініціали) (шифр)

спеціальності 123 Комп'ютерна інженерія

за освітньо-професійною програмою Комп'ютерна інженерія
(офіційна назва)

на тему " Комп'ютерна підсистема перевірки цілісності та достовірності файлів при передачі даних"

затверджену наказом ректора НТУ «Дніпровська політехніка» від 05.05.2025 № 336-с

Розділ	Зміст	Термін виконання
Загальний розділ	На основі матеріалів виробничих практик, інших науково-технічних джерел показати актуальність завдання, сформулювати мету та задачі виконання кваліфікаційної роботи	10.02.2025
Спеціальний розділ	Сформулювати найменування й призначення комп'ютерної підсистеми перевірки цілісності та достовірності файлів при передачі даних, висунути технічні вимоги до нього	15.03.2025
	Розробити архітектуру комп'ютерної підсистеми, обґрунтування технічних характеристик.	20.04.2025
	Реалізувати комп'ютерну підсистему, використовуючи відповідні інструменти програмування та враховуючи принципи розробки надійних та ефективних програмних систем, що є частиною комп'ютерної інженерії	07.05.2025
	Протестувати розроблену комп'ютерну підсистему на предмет її функціональності, продуктивності обміну даними в реальному часі та стійкості до можливих збоїв	31.05.2025

Завдання видано _____ доц. Каштан В.Ю.
(підпис керівника) (прізвище, ініціали)

Дата видачі 25.01.2025

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Субач О.А.

РЕФЕРАТ

Пояснювальна записка: 62 с., 13 рис., 1 табл., 1 додаток, 14 джерел.

Об’єкт дослідження – система захисту даних, що використовується для за шифрування і підпису даних в файлах, задля безпечної передачі цих даних по незахищеним каналам передачі інформації.

Предмет дослідження – методи та алгоритми криптографічного захисту інформації, що забезпечують перевірку цілісності та достовірності файлів, а також принципи проектування та розробки комп'ютерних підсистем для їх практичної реалізації.

Мета роботи – розробка та практична реалізація комп'ютерної підсистеми для ефективної перевірки цілісності та достовірності файлів під час їх передачі, що забезпечується використанням сучасних криптографічних алгоритмів..

Завдання:

1. Аналіз існуючих методів та моделей захисту даних.
2. Вибір оптимальних криптографічних алгоритмів.
3. Проектування архітектури комп'ютерної підсистеми.
4. Реалізація модуля шифрування та дешифрування.
5. Реалізація модуля електронного підпису.
6. Розробка інтерфейсу користувача.
7. Тестування підсистеми.
8. Оцінка ефективності та рекомендації.

Ключові слова: захист даних, відкритий ключ, симетричне шифрування.

ЗМІСТ

ВСТУП 6
1 АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАВДАННЯ 8
1.1 Захист інформації та шифрування 8
1.3 Захист даних від несанкціонованого доступу 13
1.4 Криптографічні методи захисту інформації 17
1.5 Мета і задачі 23
2 СПЕЦІАЛЬНИЙ РОЗДІЛ 26
2.1 Технічні вимоги до комп'ютерної підсистеми перевірки цілісності та достовірності файлів при передачі даних 26
2.1.1 Формування загальних вимог до підсистеми 26
2.1.2 Вимоги до структури і функціонування системи 27
2.2 Розробка апаратної частини 29
2.2.1 Архітектура комп'ютерної підсистеми перевірки цілісності та достовірності файлів при передачі даних 29
2.3 Програмування комп'ютерної підсистеми перевірки цілісності та достовірності файлів при передачі даних 32
2.3.1 Реалізація генерації ключів 32
2.3.3 Створення підпису 34
2.3.4 Збереження підпису 37
2.3.5 Робота з ключем. Зчитування з файлу і перетворення в PrivateKey 41
2.3.6 Реалізація алгоритму DES для шифрування і розшифровки файлу 43
2.4 Розробка інтерфейсу програми 45
2.5 Інструкція користувача 47
ДОДАТОК А 56

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ ТА ТЕРМІНІВ

AES – Advanced Encryption Standard (Розширений стандарт шифрування)

API – Application Programming Interface (Інтерфейс програмування застосунків)

DES – Data Encryption Standard (Стандарт шифрування даних)

DSA – Digital Signature Algorithm (Алгоритм цифрового підпису)

ECC – Elliptic Curve Cryptography (Криптографія на еліптичних кривих)

GUI – Graphical User Interface (Графічний інтерфейс користувача)

JCA – Java Cryptography Architecture (Архітектура криптографії Java)

RSA – Rivest–Shamir–Adleman (Алгоритм RSA, названий на честь розробників)

SHA – Secure Hash Algorithm (Безпечний алгоритм хешування)

SHA-1 – Secure Hash Algorithm 1 (Безпечний алгоритм хешування, версія 1)

SHA-256 – Secure Hash Algorithm 256 (Безпечний алгоритм хешування з довжиною хешу 256 біт)

X.509 – Стандарт X.509, що визначає формат публічних ключів та сертифікатів.

ВСТУП

На сьогодні в інформаційному просторі, швидкими темпами впроваджуються новітні досягнення комп'ютерних і телекомунікаційних технологій. Комп'ютерні системи активно впроваджуються у фінансові, промислові, торгові і соціальні сфери. Внаслідок цього різко зріс інтерес широкого кола користувачів до проблем захисту інформації. Захист інформації - це сукупність організаційно-технічних заходів і правових норм для попередження заподіяння збитку інтересам власника інформації. В останні роки з розвитком комерційної і підприємницької діяльності збільшилося число спроб несанкціонованого доступу (НСД) до конфіденційної інформації.

Серед всього спектру методів захисту даних від небажаного доступу особливе місце займають криптографічні методи. Криптографія - наука про математичні методи забезпечення конфіденційності і автентичності інформації. Для сучасної криптографії характерне використання відкритих алгоритмів шифрування, що припускають використання обчислювальних засобів.

Криптографічний захист інформації – вид захисту інформації,що реалізується шляхом перетворення інформації з використанням спеціальних (ключових) даних з метою приховування/відновлення змісту інформації, підтвердження її справжності, цілісності, авторства тощо. [1]

Завданням дипломної роботи є розробка програми, яка забезпечить захист даних, тобто інформації, яку ми зберігаємо на ПЗП, передаємо електронною поштою, друкуємо в текстових редакторах і т.д., від НСД.

Метою є розробка програми для шифрування та дешифрування даних за допомогою криптографічних алгоритмів. Для цього використаємо два алгоритми шифрування такі як RSA та DES.

Алгоритм DES (Data Encryption Standard) був стандартом симетричних блочних шифрів затверджених урядом США до 2001року (зараз використовується в режимі 3DES). Під словом «симетричний» розуміють те що для шифрування і дешифрування використовується один і той же ключ, а блоковий тому що шифрування даних відбувається поблоково. Тобто дані розбиваються на блоки

фіксованої довжини (як правило для DES, довжина блоку дорівнює 64 біт), а потім шифруються. Даний алгоритм використовується для великих об’ємів даних. [2]

Алгоритм RSA є асиметричним шифром (або з відкритим ключем) в якому використовується ключ який складається з двох частин: відкритий (public key), що зашифровує дані, і відповідний йому закритий (private key), що їх розшифровує. Відкритий ключ поширюється по усьому світу, у той час як закритий тримається в таємниці. Хоча ключова пара математично зв'язана, обчислення закритого ключа з відкритого в практичному плані неможлива. Кожний, у кого є відкритий ключ, зможе зашифрувати дані, але не зможе їх розшифрувати. Тільки людина, яка володіє відповідним закритим ключем, може розшифрувати інформацію. [3]

1 АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАВДАННЯ

1.1 Захист інформації та шифрування

Захист інформації (або Data protection в англійській термінології) — це комплексний набір підходів та інструментів, що застосовуються для забезпечення цілісності, конфіденційності та доступності інформації. Це стає особливо важливим, коли інформація може бути піддана впливу загроз як природного, так і штучного походження. Реалізація таких загроз здатна завдати шкоди власникам та користувачам інформації.

Серед різноманіття методів захисту даних від несанкціонованого доступу, криптографічні методи посідають особливе місце. Сучасні підходи до шифрування здатні забезпечити майже абсолютний рівень захисту даних, проте завжди існує проблема надійності їх впровадження. Наразі, особливо актуальною є задача оцінки вже існуючих криптоалгоритмів. Визначення ефективності засобів захисту інформації часто виявляється значно більш трудомістким процесом, ніж їх безпосередня розробка. Ця діяльність потребує наявності спеціалізованих знань і, як правило, вищої кваліфікації, ніж сама розробка. Такі обставини призводять до того, що на ринку з'являється велика кількість засобів криптографічного захисту інформації, про які часто бракує об'єктивної інформації. При цьому розробники нерідко тримають свої криптоалгоритми в таємниці.

Шифрування — це спеціалізований спосіб модифікації повідомлення чи іншого документа, що має на меті спотворити або приховати його зміст. Слід зазначити, що кодування відрізняється від шифрування тим, що це перетворення звичайного, зрозумілого тексту в код, яке виконується без використання ключа. Зашифрувати можна не лише текст, але й різноманітні комп'ютерні файли — від файлів баз даних та текстових процесорів до файлів зображень.

Основна ідея шифрування полягає в запобіганні доступу до справжнього змісту повідомлення (тексту, файлу тощо) для тих, хто не володіє засобами для його дешифрування. Таким чином, прочитати зашифрований файл зможе лише та особа, яка матиме можливість його дешифрувати.

Шифрування виникло приблизно чотири тисячі років тому. Першим відомим випадком використання шифру (коду) вважається єгипетський текст, датований приблизно 1900 роком до нашої ери. Автор цього тексту використовував знаки, що не відповідали звичайним (для єгиптян) ієрогліфам.

Одним з найвідоміших методів шифрування є той, що носить ім'я Цезаря. Хоча, можливо, він і не був його винахідником, він активно його застосовував. Не довіряючи своїм посланцям, Цезар шифрував листи за допомогою елементарної заміни: A на D, B на E і так далі по всьому латинському алфавіту. При такому кодуванні комбінація XYZ була б записана як ABC (прямий код N+3) [4].

1.2 Основи безпеки даних в комп'ютерних системах

Витік комерційної інформації призводить до небажаних наслідків, а збитки, спричинені діяльністю конкурентів, що використовують методи шпигунства, можуть бути надзвичайно великими. Тому питання безпеки, у найрізноманітніших її проявах, доводиться вирішувати щоразу при розгляді будь-яких аспектів людської діяльності. Важливо розуміти, що всі види безпеки тісно пов'язані з інформаційною безпекою (ІБ), і, більше того, їх неможливо забезпечити без належного рівня ІБ.

Інформація — це сукупність відомостей про осіб, факти, предмети, події, явища та процеси, незалежно від форми їх представлення.

Захист інформації — це комплекс заходів, що здійснюються з метою запобігання (або зниження до безпечного рівня) можливостей витоку, розкрадання, втрати, поширення, знищення, перекручування, підробки або блокування інформації [5].

Види дій, що можуть бути здійснені над інформацією:

– блокування інформації: ситуація, коли користувач не може отримати доступ до інформації, проте сама інформація при цьому не втрачається;

– порушення цілісності: включає втрату або вихід з ладу носія інформації, спотворення (тобто порушення смислової значущості), порушення логічної зв'язаності, а також втрату достовірності (коли наявна інформація не відповідає реальному стану речей);

– порушення конфіденційності: відбувається, коли з інформацією ознайомлюються суб'єкти, на яких це не покладено. рівень доступу до інформації визначається її власником. порушення конфіденційності може бути наслідком некоректної роботи системи обмеження доступу або наявності побічного каналу доступу;

– несанкціоноване тиражування: в цьому контексті захист передбачає захист авторських прав та прав власності на інформацію.

Автоматизована система (АС) — це організаційно-технічна система, що об'єднує обчислювальну систему, фізичне середовище, персонал та оброблювану інформацію [5].

Захист інформації в АС (також відомий як information security або computer system security) — це діяльність, спрямована на забезпечення безпеки інформації, що обробляється в АС, а також АС в цілому. Вона дозволяє запобігти або ускладнити можливість реалізації загроз, а також зменшити розмір потенційних збитків від їх реалізації.

Комплексна система захисту інформації (КСЗІ) — це сукупність організаційних та інженерних заходів, програмно-апаратних засобів, які забезпечують захист інформації в АС.

Загроза — це потенційно можлива подія, дія, процес або явище, яке може призвести до нанесення шкоди інтересам певної фізичної чи юридичної особи. Реалізацією загрози є порушення роботи системи. Загрози поділяються на природні та штучні.

Природні загрози — це загрози, спричинені дією на АС об'єктивних фізичних процесів або стихійних природних явищ, незалежних від людини. До них належать: стихійні лиха, магнітні бурі, радіоактивне випромінювання, опади тощо, а також загрози опосередковано технічного характеру, пов'язані з надійністю технічних засобів обробки інформації та підсистем забезпечення АС.

Штучні загрози — це загрози, викликані діяльністю людини. Вони поділяються на:

– ненавмисні — загрози, пов'язані з випадковими діями людей,

спричиненими незнанням, халатністю, цікавістю, але без злого наміру;

– навмисні — дії людини, що здійснюються умисно для дезорганізації роботи системи, виведення її з ладу, для незаконного проникнення в систему та несанкціонованого доступу до інформації.

Система складових загроз безпеки даних представлена в Таблиці 1.1 (припускаючи, що ця таблиця існує в оригінальному документі).

Таблиця 1.1 Класифікаційні складові загроз безпеки інформації

Параметр класифікації	Значення параметра	Зміст
1. Види	1.1. Фізична цілісність	- знищення (спотворення);
	1.2. Логічна цілісність	- спотворення;
	1.3. Конфіденційність	- несанкціоноване отримання;
	1.4. Порушення прав власності	- привласнення чужого права
2.Природа походження	2.1.Випадкова 2.2.Навмисна	-відмови, збої, помилки,стихійні біди; -зловмисні дії людей
3. Передумови появи	3.1. Об'єктивні 3.2. Суб'єктивні	-кількісна або якісна
		недостатність елементів систем;
		-розвідувальні органи іноземної держави
4. Джерела загрози	4.1. Люди 4.2. Технічні пристрої 4.3. ПЗ (ППЗ, СМЗ) 4.4. Зовнішнє середовище	-сторонній персонал; -пристрої обробки, зберігання, передачі інформації; -помилки; -атмосфера, побічні явища

Сучасні засоби перехоплення інформації дозволяють на відстані в десятки та сотні, а іноді й більше метрів реєструвати різної природи побічні інформативні сигнали, що виникають під час роботи технічних засобів. За результатами цієї реєстрації можна відновлювати інформацію, що обробляється, передається, приймається або копіюється [6].

Інформацію можна отримати не тільки шляхом перехоплення побічних інформативних сигналів, але й за результатами прямої реєстрації сигналів, що циркулюють в інформаційних ланцюгах технічних систем (насамперед, у лініях зв'язку). Як правило, реалізувати засоби перехоплення в цьому випадку легше, ніж у випадку побічних випромінювань та наведень.

Фізичні заходи захисту інформації базуються на застосуванні різноманітних механічних, електро- або електронно-механічних пристроїв. Ці пристрої спеціально призначені для створення фізичних перешкод на можливих шляхах проникнення та доступу потенційних порушників до компонентів системи та інформації. Крім того, вони включають технічні засоби візуального нагляду, зв'язку та охоронної сигналізації.

Ідентифікація — це процес присвоєння унікального ідентифікатора суб'єктам або об'єктам доступу, або ж порівняння наданого ідентифікатора з уже існуючим переліком присвоєних ідентифікаторів. Якщо говорити про загальне розуміння, ідентифікація об'єкта означає його розпізнавання або ототожнення з чимось. У сфері інформаційних технологій цей термін зазвичай стосується встановлення особистості користувача. Цей процес є вкрай важливим для того, щоб система могла надалі вирішити, чи надавати людині дозвіл на роботу з комп'ютером, доступ до закритої інформації тощо. Отже, ідентифікація є одним із фундаментальних понять в інформаційній безпеці.

Аутентифікація — це процедура перевірки, яка підтверджує належність ідентифікатора певному суб'єкту. Цей процес базується на використанні певного секретного елемента, який називається аутентифікатором. Цей аутентифікатор має бути доступним як для суб'єкта (користувача), так і для інформаційної системи. Зазвичай інформаційна система не зберігає сам секретний елемент у чистому вигляді, а має лише певну інформацію про нього, на основі якої приймається рішення щодо відповідності суб'єкта заявленому ідентифікатору. Наприклад, перед початком інтерактивного сеансу роботи більшість операційних систем вимагають від користувача ввести його ім'я та пароль. У цьому випадку введене ім'я є ідентифікатором користувача, а його пароль — аутентифікатором. Важливо

зазначити, що операційна система зазвичай зберігає не сам пароль, а його хеш-суму, що суттєво ускладнює можливість його відновлення.

1.3 Захист даних від несанкціонованого доступу

Одним із ключових напрямків захисту інформації в сучасних інформаційних системах є технічний захист інформації (ТЗІ). Це широке поле діяльності, яке поділяється на два основні класи завдань: захист інформації від несанкціонованого доступу (НСД) та захист інформації від витоку технічними каналами. Під НСД розуміється будь-який доступ до інформації, який порушує встановлену в інформаційній системі політику розмежування доступу. У свою чергу, технічні канали витоку інформації включають різноманітні шляхи, такі як побічні електромагнітні випромінювання та наведення (наприклад, випромінювання від кабелів чи електронних компонентів, які можна перехопити), акустичні канали (звуки, що виходять від пристроїв або розмов), оптичні канали (випромінювання від світлодіодів, моніторів) та інші неочевидні вектори, через які дані можуть бути скомпрометовані [6].

Захист від НСД може бути реалізований на різних рівнях та в різних складових частинах інформаційної системи, що забезпечує багатошаровий підхід до безпеки.

Прикладне та системне програмне забезпечення (ПЗ) це включає операційні системи, офісні програми, системи управління базами даних, спеціалізовані бізнес-додатки та інше програмне забезпечення, яке обробляє, зберігає або передає інформацію. Захист тут спрямований на запобігання злому, ін'єкціям коду та іншим програмним вразливостям.

Апаратна частина серверів та робочих станцій: Це фізичні компоненти комп'ютерів, що зберігають та обробляють дані, такі як центральні процесори, пам'ять, жорсткі диски та інтерфейси. Захист на цьому рівні передбачає убезпечення від несанкціонованого фізичного доступу та маніпуляцій з обладнанням.

Комунікаційне обладнання та канали зв'язку це мережеві пристрої

(маршрутизатори, комутатори, точки доступу Wi-Fi) та лінії передачі даних (дротові й бездротові), через які відбувається обмін інформацією. Захист тут фокусується на шифруванні трафіку та запобіганні перехопленню даних під час їх передачі.

Периметр інформаційної системи: Зовнішні межі мережі та фізичного простору, що захищають внутрішні ресурси від зовнішніх загроз. Це може включати фізичні бар'єри, системи контролю доступу до будівель, а також мережеві шлюзи та граничні маршрутизатори.

Для кожного з вищезазначених напрямків існують спеціалізовані засоби захисту, які працюють спільно для створення комплексної системи безпеки.

Для захисту інформації на рівні прикладного та системного ПЗ використовуються:

– системи розмежування доступу до інформації: ці системи визначають, які користувачі або процеси мають право на доступ до певних ресурсів та які операції вони можуть виконувати (читання, запис, виконання, видалення). це реалізується через права доступу до файлів, папок, баз даних тощо;

– системи ідентифікації та аутентифікації: вони забезпечують перевірку особистості користувача (наприклад, за допомогою логіну та паролю, біометричних даних, смарт-карток або багатофакторної аутентифікації) перед наданням доступу до системи;

– системи аудиту та моніторингу: ці системи реєструють всі значущі події в системі, дозволяючи відстежувати дії користувачів, зміни в системі та виявляти аномалії, що можуть свідчити про спробу нсд або порушення політики безпеки;

– системи антивірусного захисту: захищають від шкідливого програмного забезпечення (вірусів, троянів, програм-вимагачів, шпигунського пз), яке може бути використане для крадіжки інформації, порушення роботи системи або отримання несанкціонованого доступу.

Для захисту інформації на рівні апаратного забезпечення використовуються:

– апаратні ключі: фізичні пристрої (наприклад, usb-токени, смарт-карти), що служать для надійної аутентифікації користувачів, шифрування даних або зберігання криптографічних ключів;

– системи сигналізації: забезпечують виявлення несанкціонованого проникнення до фізичних об'єктів (серверних кімнат, офісів), де розміщена інформація або обладнання;

– засоби блокування пристроїв та інтерфейсів вводу-виводу інформації: обмежують можливість несанкціонованого підключення зовнішніх пристроїв (usb-накопичувачів, мобільних телефонів) або використання портів для копіювання даних на незахищені носії.

У комунікаційних системах для мережевого захисту інформації застосовуються:

– міжмережеві екрани (firewall): контролюють вхідний та вихідний мережевий трафік, блокуючи небажані з'єднання або пакети даних згідно з встановленими правилами безпеки, діючи як бар'єр між внутрішньою та зовнішньою мережами;

– системи виявлення вторгнень (ids - intrusion detection system) та системи запобігання вторгненням (ips - intrusion prevention system): відстежують мережеву активність на предмет ознак вторгнень або спроб атак (наприклад, сканування портів, спроби перебору паролів), сповіщаючи про них адміністраторів або автоматично блокуючи шкідливу активність;

– засоби створення віртуальних приватних мереж (vpn - virtual private network): створюють зашифровані тунелі для безпечної передачі даних через незахищені мережі, такі як інтернет, забезпечуючи конфіденційність та цілісність інформації навіть у громадських мережах;

– засоби аналізу захищеності: інструменти для проведення тестів на проникнення (пентестів), сканування вразливостей, аудиту конфігурацій та оцінки загального рівня захищеності мережевої інфраструктури.

Для захисту периметра інформаційної системи створюються:

– системи охоронної та пожежної сигналізації: забезпечують фізичний захист об'єктів та приміщень, де розміщена інформація, від несанкціонованого доступу, крадіжок чи надзвичайних ситуацій;

– системи цифрового відеоспостереження: дозволяють візуально контролювати доступ до об'єктів, фіксувати потенційно небезпечні події та надавати доказову базу у разі інцидентів;

– системи контролю та керування доступом (сккд): регулюють доступ людей до певних зон (будівель, приміщень, серверних), використовуючи електронні перепустки, біометричні дані (відбитки пальців, розпізнавання обличчя) або інші методи.

При розробці програмно-апаратних засобів захисту від несанкціонованого доступу керуються такими ключовими принципами, що формують основу надійної архітектури безпеки:

– принцип обґрунтованості доступу (принцип найменших привілеїв): кожен виконавець, користувач або системний процес повинен мати лише той рівень доступу та до тієї закритої інформації, яка абсолютно необхідна для повноцінного та ефективного виконання його безпосередніх професійних обов'язків. це мінімізує потенційні збитки у випадку компрометації облікового запису;

– принцип достатньої глибини контролю доступу: системи захисту інформації (сзі) повинні включати комплексні та багаторівневі механізми контролю доступу до всіх без винятку видів інформаційних та програмних ресурсів, включаючи файли, каталоги, бази даних, системні реєстри, мережеві порти та програмні функції;

– принцип розмежування потоків інформації: забороняє переписування закритої інформації на незахищені носії або передачу її через незахищені канали. це часто досягається шляхом маркування (класифікації) інформації та її носіїв, а також встановлення суворих правил передачі;

– принцип чистоти повторно використовуваних ресурсів: перед повторним використанням системних ресурсів (наприклад, дискового простору, оперативної пам'яті), які раніше містили конфіденційну або закриту інформацію, вони мають бути повністю та надійно очищені від неї, щоб запобігти витоку даних;

– принцип персональної відповідальності (незаперечності): кожен виконавець несе персональну відповідальність за всю свою діяльність у системі,

включаючи всі дії із закритою інформацією. це вимагає впровадження ефективних механізмів аудиту, журналювання та забезпечення неможливості відмови від скоєних дій (non-repudiation);

– принцип цілісності засобів захисту: засоби захисту повинні функціонувати точно згідно з їхнім призначенням, без збоїв чи вразливостей, і бути максимально ізольованими від прямого втручання або модифікації з боку користувача чи інших процесів, щоб уникнути їх компрометації [7].

Не слід недооцінювати можливості непрофесіоналів у здійсненні комп'ютерних злочинів. Часто, на жаль, нелояльні співробітники, які мають законний доступ до комп'ютерів та внутрішньої інформації, відіграють головну роль у більшості фінансових злочинів та витоків даних. Це є скоріше організаційною та кадровою проблемою, ніж суто технічною. Хоча процедури безпеки можуть забезпечувати сувору перевірку паролів та контроль доступу до цінних загальних даних, зловмисника, обізнаного у внутрішньому устрої системи, практично неможливо повністю зупинити лише технічними засобами. Саме тому для побудови надійного та ефективного захисту необхідно застосовувати комплексний підхід: ретельно виявити всі можливі загрози безпеці інформації, об'єктивно оцінити їх потенційні наслідки, визначити необхідні заходи та засоби захисту (як технічні, так і організаційні), і постійно оцінювати їх ефективність, адаптуючись до нових викликів та вразливостей [8].

1.4 Криптографічні методи захисту інформації

Криптографічний захист інформації є фундаментальним видом захисту, що реалізується через перетворення даних із застосуванням спеціальних ключових даних. Його основна мета — приховати (або, навпаки, відновити) зміст інформації, а також підтвердити її справжність, цілісність та авторство [1]. В основі цього захисту лежить криптографія (від грецького kryptos — прихований і graphein — писати) — наука, що вивчає математичні методи забезпечення конфіденційності (неможливості прочитання інформації сторонніми особами) та автентичності (підтвердження

цілісності та справжності авторства) інформації. Криптографія розвинулася з давньої практичної потреби безпечної передачі важливих відомостей і активно використовує інструментарій абстрактної алгебри для математичного аналізу [20].

Сучасна криптографія вирізняється використанням відкритих алгоритмів шифрування, які ефективно працюють з обчислювальними засобами. Існує понад десяток перевірених алгоритмів шифрування, які при використанні достатньо довгого ключа та коректної реалізації алгоритму роблять зашифрований текст практично недоступним для криптоаналізу. Серед широко використовуваних алгоритмів можна назвати Twofish, IDEA, RC4 та інші. Багато країн розробили власні національні стандарти шифрування. Наприклад, у 2001 році в США був прийнятий стандарт симетричного шифрування AES (Advanced Encryption Standard), що базується на алгоритмі Rijndael з довжиною ключа 128, 192 або 256 біт. AES прийшов на зміну колишньому стандарту DES (Data Encryption Standard), який тепер рекомендується використовувати лише в режимі Triple-DES (3DES) для підвищення безпеки.

Тривалий час під криптографією розумілося виключно шифрування — процес перетворення звичайної, зрозумілої інформації (так званого відкритого тексту) у незрозуміле «сміття» (шифротекст). Дешифрування, відповідно, є зворотним процесом відновлення вихідної інформації із шифротексту. Пара алгоритмів шифрування та дешифрування називається шифром. Дія шифру керується не лише алгоритмами, але й у кожному конкретному випадку певним ключем [9].

Ключ — це секретний параметр, який в ідеалі відомий лише двом сторонам, що беруть участь у передачі повідомлення, і використовується для окремого контексту зв'язку. Ключі мають величезне значення, оскільки без змінних ключів алгоритми шифрування легко зламуються і стають непридатними для використання в більшості випадків. Історично склалося так, що шифри часто використовувалися лише для шифрування та дешифрування, без виконання додаткових процедур, таких як автентифікація або перевірка цілісності повідомлення.

В англійській мові терміни "cryptography" (криптографія) та "cryptology" (криптологія) іноді використовуються як синоніми. Проте, часто під криптографією

розуміється безпосередньо використання та дослідження технологій шифрування, тоді як криптологія є ширшою галуззю, що охоплює як криптографію, так і криптоаналіз. Криптоаналіз — це розділ криптології, що займається математичними методами порушення конфіденційності та цілісності інформації без знання ключа. Отже, криптографія займається розробкою методів шифрування даних, тоді як криптоаналіз зосереджений на оцінці сильних і слабких сторін цих методів, а також на розробці шляхів для злому криптосистем [8]. Дослідження характеристик мов, що мають відношення до криптології (таких як частоти появи певних літер, комбінацій літер, загальні шаблони тощо), називається криптолінгвістикою.

До недавнього часу криптографія займалася виключно забезпеченням конфіденційності повідомлень (тобто шифруванням) — перетворенням інформації зі зрозумілої форми в незрозумілу і зворотним її відновленням на стороні одержувача. Це робило повідомлення неможливим для прочитання для того, хто перехопив або підслухав, не маючи секретного знання (а саме, ключа, необхідного для дешифрування повідомлення). Проте в останні десятиліття сфера застосування криптографії значно розширилася. Вона тепер включає не лише таємну передачу повідомлень, але й такі важливі аспекти, як методи перевірки цілісності повідомлень, ідентифікації відправника/одержувача (аутентифікація), створення цифрових підписів, інтерактивні підтвердження та розробку загальних технологій безпечного спілкування, що забезпечують довіру та надійність у цифровому середовищі [7].

Найперші форми тайнопису вимагали не більше ніж аналог олівця та паперу, оскільки в ті часи більшість людей не могли читати. Поширення писемності, особливо серед ворогів, викликало нагальну потребу саме в криптографії. Основними типами класичних шифрів були перестановочні шифри, які змінювали порядок літер у повідомленні, та підстановочні шифри, що систематично замінювали літери або групи літер іншими символами чи групами. Прості варіанти обох типів пропонували слабкий захист від досвідчених супротивників. Одним із ранніх підстановочних шифрів був шифр Цезаря, в якому кожна літера в повідомленні замінювалась літерою через декілька позицій із абетки. Цей шифр

отримав ім'я Юлія Цезаря, який його активно використовував зі зсувом у 3 позиції для спілкування з генералами під час військових кампаній, подібно до коду EXCESS-3 у булевій алгебрі.

Шляхом застосування шифрування намагалися зберегти зміст спілкування в таємниці, подібно до шпигунів, військових лідерів та дипломатів. Збереглися також відомості про деякі з ранніх єврейських шифрів. Застосування криптографії навіть радиться в Камасутрі як спосіб спілкування закоханих без ризику незручного викриття. Стеганографія (тобто, приховування самого факту наявності повідомлення) також була розроблена в давні часи. Зокрема, Геродот описав випадок, коли повідомлення було приховане у вигляді татуювання на поголеній голові раба, а потім прикрите відрослим волоссям. До сучасних прикладів стеганографії належать використання невидимих чорнил, мікрокрапок, а також цифрових водяних знаків, що застосовуються для приховування інформації в медіафайлах [3].

Шифротексти, отримані від класичних шифрів (та деяких менш досконалих сучасних), завжди видають певну статистичну інформацію про вихідний текст повідомлення, яка може бути використана для зламу. Після відкриття частотного аналізу (ймовірно, арабським вченим аль-Кінді) у IX столітті, майже всі такі шифри стали більш-менш легко зламними для досвідченого фахівця. Класичні шифри зберегли популярність переважно у вигляді головоломок. Майже всі шифри залишались беззахисними перед криптоаналізом з використанням частотного аналізу до винаходу поліалфавітного шифру, швидше за все, Леоном-Баттістою Альберті приблизно у 1467 році (хоча існують свідчення, що знання про такі шифри існували серед арабських вчених). Винахід Альберті полягав у використанні різних шифрів (наприклад, алфавітів підстановки) для різних частин повідомлення. Йому також належить винахід того, що можна вважати першим шифрувальним приладом: колеса, яке частково реалізовувало його ідею. У поліалфавітному шифрі Віженера (Vigenère cipher) алгоритм шифрування використовує ключове слово, яке керує підстановкою літер залежно від того, яка літера ключового слова використовується. Проте, у середині 1800-х років Чарльз Беббідж показав, що поліалфавітні шифри

цього типу також залишалися частково вразливими перед частотним аналізом [4].

Хоча частотний аналіз є потужною та загальною технікою, шифрування на практиці часто було ефективним, оскільки багато криптоаналітиків не володіли цією технікою. Дешифрування повідомлень без частотного аналізу практично означало необхідність знання використаного шифру, що спонукало до шпигунства, підкупу, крадіжок, зрад тощо для отримання алгоритму. Згодом, у XIX столітті, було визнано, що збереження алгоритму шифрування в таємниці не забезпечує захист від зламу; насправді, було встановлено, що будь-яка адекватна криптографічна схема залишається у безпеці, навіть за умови доступу сторонніх. Збереження в таємниці лише ключа має бути достатньою умовою захисту інформації нормальним шифром. Цей фундаментальний принцип був вперше проголошений у 1883 році Огюстом Керкгофсом і загальновідомий як принцип Керкгоффса: безпека криптосистеми не повинна залежати від збереження її алгоритму в таємниці, а лише від секретності ключа. Більш різкий варіант озвучив Клод Шеннон як максимум Шеннона: ворог знає систему.

Було створено різні механічні прилади та інструменти для допомоги в шифруванні. Одним з найперших є скітала у стародавній Греції — палиця, що, як вважається, використовувалась Спартанцями як перестановочний шифр. У середньовіччі було винайдено інші засоби, такі як дірочний шифр, що також використовувався для часткової стеганографії. Разом із винаходом поліалфавітних шифрів були розроблені досконаліші засоби, такі як власний винахід Альберті — шифрувальний диск та мультициліндр Томаса Джефферсона (повторно винайдений Базерієсом приблизно у 1900 році). Кілька механічних шифрувально-дешифрувальних приладів було створено на початку XX століття і багато запатентовано, серед них роторні машини — найвідомішою серед них є Енігма, автомат, що використовувався Німеччиною з кінця 1920-х до кінця Другої світової війни [9].

Цей автомат реалізовував складний електромеханічний поліалфавітний шифр для захисту таємних повідомлень. Злам шифру Енігми в Бюро Шифрів (Biuro Szyfrow) та, як наслідок, дешифрування повідомлень у Блетчлі Парк (Bletchley Park),

було важливим чинником перемоги Союзників у війні [8].

Шифри, реалізовані покращеними варіантами цих схем, призвели до істотного підвищення криптоаналітичної складності після Другої світової війни. Поява цифрових комп'ютерів та електроніки після Другої світової війни уможливила створення значно складніших шифрів. Більше того, комп'ютери дозволяли шифрувати будь-які дані, які можна представити в двійковому вигляді, на відміну від класичних шифрів, що розроблялися для шифрування письмових текстів. Це зробило лінгвістичні підходи в криптоаналізі непридатними для застосування. Багато комп'ютерних шифрів можна характеризувати за їхньою роботою з послідовностями бінарних бітів (інколи в блоках або групах), на відміну від класичних та механічних схем, які зазвичай працюють безпосередньо з літерами. Однак, комп'ютери також знайшли застосування у криптоаналізі, що певною мірою компенсувало підвищення складності шифрів. Тим не менше, гарні сучасні шифри залишаються попереду криптоаналізу; як правило, використання якісних шифрів дуже ефективне (тобто швидке і вимагає небагато ресурсів), тоді як злам цих шифрів потребує набагато більших зусиль, ніж раніше, що робить криптоаналіз настільки неефективним та непрактичним, що злам стає практично неможливим.

Широкі академічні дослідження криптографії з'явилися порівняно недавно — починаючи з середини 1970-х років, разом із появою відкритої специфікації стандарту DES (Data Encryption Standard) Національного Бюро Стандартів США, публікацій Діффі-Хелмана та оприлюдненням алгоритму RSA. Відтоді криптографія перетворилася на загальнопоширений інструмент для безпечної передачі даних у комп'ютерних мережах та захисту інформації загалом. Сучасний рівень безпеки багатьох криптографічних методів базується на обчислювальній складності певних математичних проблем, таких як розклад цілих чисел на прості множники або проблеми з дискретними логарифмами. У багатьох випадках існують докази безпечності криптографічних методів лише за умови неможливості ефективного розв'язання певної обчислювальної проблеми. За одним суттєвим винятком — схема одноразових блокнотів (one-time pad), яка теоретично є абсолютно безпечною, але має практичні обмеження [4].

Разом із пам'яттю про історію криптографії, розробники криптографічних алгоритмів та систем також мають брати до уваги майбутній поступ технологій у своїх розробках. Наприклад, постійне підвищення обчислювальної потужності комп'ютерів розширило поле для атак грубої сили (перебору всіх можливих ключів). Тому відповідно оновлюються стандарти щодо вибору довжини ключа. Можливі наслідки розвитку квантових комп'ютерів вже враховуються деякими розробниками криптографічних систем; анонсована поява малих реалізацій цих комп'ютерів робить важливою попередню підготовку до постквантової криптографії.

Взагалі кажучи, до початку XX століття криптографія, в основному, була пов'язана з лінгвістичними схемами. Після того, як основний акцент було зміщено, зараз криптографія інтенсивно використовує складний математичний апарат, включно з теорією інформації, теорією обчислювальної складності, статистикою, комбінаторикою, абстрактною алгеброю та теорією чисел. Криптографія є також відгалуженням інженерії, але не звичним, оскільки вона має справу з активним, розумним та винахідливим супротивником; більшість інших видів інженерних наук мають справу з нейтральними силами природи. Існують також активні дослідження щодо взаємозв'язків між криптографічними проблемами та квантовою фізикою [9].

1.5 Мета і задачі

Метою роботи є розробка функціональної комп'ютерної підсистеми, здатної забезпечити надійний захист файлів під час передачі даних через незахищені канали зв'язку. Це буде досягнуто шляхом інтеграції механізмів шифрування та дешифрування даних з використанням асиметричної криптографії (формування відкритого ключа), а також реалізації електронного підпису файлів із застосуванням закритого ключа. Підсистема має ефективно перевіряти цілісність (незмінність) та автентичність (справжність авторства) переданих файлів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Аналіз існуючих методів та моделей захисту даних. Провести детальне дослідження сучасних криптографічних алгоритмів шифрування (симетричних та

асиметричних) та алгоритмів формування електронного підпису. Особливу увагу приділити їхній ефективності, криптостійкості та застосовності для забезпечення конфіденційності та цілісності файлів.

2. Вибір оптимальних криптографічних алгоритмів. На основі проведеного аналізу обґрунтувати вибір конкретних алгоритмів шифрування (наприклад, AES для симетричного шифрування даних та RSA або Elliptic Curve Cryptography (ECC) для асиметричного шифрування ключів та формування підписів) та алгоритму хешування (наприклад, SHA-256) для забезпечення надійного захисту файлів.

3. Проектування архітектури комп'ютерної підсистеми. Розробити логічну структуру підсистеми, визначивши основні модулі (модуль шифрування/дешифрування, модуль формування/перевірки електронного підпису, модуль управління ключами) та їхню взаємодію.

4. Реалізація модуля шифрування та дешифрування. Розробити програмний модуль, що забезпечує перетворення вихідних файлів у зашифрований формат та їх зворотне відновлення. Цей модуль має підтримувати генерацію та використання відкритого ключа для обміну ключами або для шифрування даних.

5. Реалізація модуля електронного підпису. Створити програмний модуль, який дозволяє формувати електронний підпис файлів за допомогою закритого ключа відправника. Додатково розробити функціонал для верифікації цього підпису з використанням відкритого ключа відправника на стороні отримувача, що підтверджує цілісність та автентичність файлу.

6. Розробка інтерфейсу користувача. Створити зручний та інтуїтивно зрозумілий інтерфейс користувача для взаємодії з підсистемою, що дозволить легко обирати файли для обробки, виконувати операції шифрування/дешифрування та електронного підпису.

7. Тестування підсистеми. Провести комплексне тестування розробленої підсистеми, включаючи функціональне тестування, тестування продуктивності та тестування безпеки. Перевірити коректність роботи всіх модулів, швидкість обробки файлів та надійність реалізованих криптографічних функцій.

8. Оцінка ефективності та рекомендації. Проаналізувати отримані

результати, оцінити відповідність розробленої підсистеми заявленим вимогам щодо безпеки та функціональності. Сформулювати рекомендації щодо можливих покращень, масштабування або подальшого розвитку системи.

2 СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1 Технічні вимоги до комп'ютерної підсистеми перевірки цілісності та достовірності файлів при передачі даних

2.1.1 Формування загальних вимог до підсистеми

Розроблювана комп'ютерна підсистема призначена для забезпечення конфіденційності, цілісності та достовірності файлів під час їх передачі через потенційно незахищені канали зв'язку. До підсистеми висуваються наступні загальні технічні вимоги. Вона повинна забезпечувати шифрування та дешифрування файлів будь-якого типу (текстові документи, зображення, архіви тощо), надійно приховуючи їхній зміст та коректно відновлюючи оригінал. Критичним функціоналом є генерація асиметричних ключів (пари відкритого та закритого ключів) для кожного користувача, а також можливість їх безпечного збереження та подальшого завантаження. Підсистема також зобов'язана дозволяти формування електронного підпису файлу за допомогою закритого ключа відправника та його подальшу верифікацію за допомогою відкритого ключа отримувача. Це необхідно для підтвердження цілісності файлу (виявлення будь-яких змін після підписання) та його автентичності (підтвердження особи відправника).

Щодо надійності, реалізовані криптографічні алгоритми та протоколи повинні відповідати сучасним стандартам криптостійкості, гарантуючи стійкість до відомих атак, таких як атаки грубої сили, частотний аналіз чи атаки на реалізацію. Підсистема має гарантувати цілісність даних протягом усіх операцій та коректно обробляти помилки (наприклад, некоректний ключ, пошкоджений файл), інформуючи користувача та запобігаючи несанкціонованому доступу. Продуктивність є ключовим аспектом: операції шифрування/дешифрування та формування/перевірки підпису мають виконуватися з прийнятною швидкістю для файлів типових розмірів, а сама підсистема повинна ефективно використовувати системні ресурси (процесорний час, оперативна пам'ять), особливо при роботі з великими файлами.

Інтерфейс користувача має бути інтуїтивно зрозумілим і простим у використанні, не вимагаючи глибоких знань у криптографії. Користувач повинен отримувати чіткі та інформативні повідомлення про статус операцій, можливі помилки та результати перевірки. Важливо передбачити зручні засоби для вибору файлів та каталогів для обробки. Вимоги до безпеки є пріоритетними: ключові дані (особливо закриті ключі) повинні зберігатися та оброблятися в безпечному режимі, мінімізуючи ризик їх витоку. Криптографічні операції мають виконуватися в ізольованому середовищі, а підсистема не повинна містити прихованих можливостей чи вразливостей, що дозволяють обійти захисні механізми ("бекдорів"). Нарешті, для забезпечення супроводжуваності, архітектура підсистеми повинна бути модульною для спрощення подальшого розширення функціоналу або заміни компонентів, а також має бути розроблена вичерпна технічна документація, що описує її архітектуру, використані алгоритми, принципи функціонування та інструкції з використання [7].

2.1.2 Вимоги до структури і функціонування системи

Підсистема має бути реалізована як модульна архітектура, що складається з наступних ключових компонентів:

– модуль управління ключами: відповідає за генерацію, зберігання та керування криптографічними ключами (як асиметричними, так і, за необхідності, симетричними). він повинен забезпечувати безпечне генерування пар відкритих/закритих ключів, їхнє надійне зберігання (можливо, із застосуванням шифрування ключів або інтеграції з апаратними модулями безпеки, hsm), а також операції експорту/імпорту ключів у захищеному форматі.

– модуль шифрування/дешифрування відповідає за виконання криптографічних перетворень даних. він повинен реалізовувати обрані алгоритми шифрування (наприклад, aes для вмісту файлу та rsa/ecc для захисту симетричного ключа) та забезпечувати як шифрування, так і дешифрування файлів.

– модуль формування/перевірки електронного підпису: цей модуль

відповідає за створення хеш-функції файлу (наприклад, sha-256), її подальше шифрування закритим ключем відправника для формування електронного підпису, а також за зворотний процес верифікації підпису за допомогою відкритого ключа. Він має забезпечувати виявлення будь-яких, навіть незначних, змін у файлі після підписання.

– модуль файлового вводу/виводу відповідає за коректне читання вихідних файлів та запис оброблених (зашифрованих, дешифрованих, підписаних) файлів. він повинен забезпечувати підтримку різних файлових форматів та великих розмірів файлів.

– модуль взаємодії з користувачем (інтерфейс) надає графічний або командний інтерфейс для взаємодії з підсистемою. Він має забезпечувати інтуїтивно зрозуміле управління всіма функціями: вибір файлів, запуск операцій, відображення стану та результатів, управління ключами.

– модуль журналювання та аудиту відповідає за фіксацію всіх значущих подій у системі, таких як генерація ключів, спроби доступу, операції шифрування/дешифрування, формування/перевірки підписів, а також будь-які виявлені аномалії або помилки. це забезпечує можливість подальшого аналізу та розслідування інцидентів.

Функціонування підсистеми має відповідати наступним вимогам:

– процеси шифрування та підписання, а також дешифрування та перевірки підпису, повинні виконуватися у логічній послідовності. Наприклад, для передачі файлу: спочатку формується електронний підпис, потім файл (або файл разом з підписом) шифрується. При отриманні: спочатку дешифрування, потім перевірка підпису;

– підсистема повинна мати надійні механізми обробки помилок, що виникають під час криптографічних операцій (наприклад, невірний ключ, пошкоджений шифротекст, невідповідність хеш-сум). У разі помилки користувачу повинно бути надано чітке повідомлення про причину та можливі дії;

– доступ до закритих ключів повинен бути суворо обмежений та захищений. Це може включати парольний захист для доступу до сховища ключів, а

також механізми контролю доступу на рівні операційної системи.

Підсистема повинна підтримувати різні режими роботи:

– шифрування файлу (з можливістю підпису) – вихідний файл шифрується з використанням відкритого ключа одержувача (або симетричного ключа, захищеного відкритим ключем). Додатково може бути створений електронний підпис файлу закритим ключем відправника;

– дешифрування файлу (з можливістю перевірки підпису). Зашифрований файл дешифрується закритим ключем одержувача. Якщо файл містить підпис, він повинен бути перевірений за допомогою відкритого ключа відправника;

– створення електронного підпису для файлу без його шифрування;

– верифікація електронного підпису для файлу без його дешифрування;

– запуск процесу генерації нової пари асиметричних ключів;

– за можливості, підсистема повинна надавати опції для автоматизації рутинних операцій, наприклад, пакетне шифрування або підписання кількох файлів;

– всі криптографічні операції повинні бути сумісними зі стандартними реалізаціями обраних алгоритмів для забезпечення взаємодії з іншими системами, що використовують ті ж стандарти.

2.2 Розробка апаратної частини

2.2.1 Архітектура комп'ютерної підсистеми перевірки цілісності та достовірності файлів при передачі даних

Для забезпечення надійного захисту файлів під час їх передачі через незахищені канали зв'язку, комп'ютерна підсистема будується за модульним принципом. Така архітектура забезпечує гнучкість, масштабованість, легкість у підтримці та високий рівень безпеки за рахунок чіткого розподілу відповідальності між компонентами. В основі архітектури лежить взаємодія ключових модулів, які виконують специфічні криптографічні та допоміжні функції.

Підсистема складається з наступних основних модулів, кожен з яких виконує свою роль у процесі обробки та захисту файлів (рис.2.1).

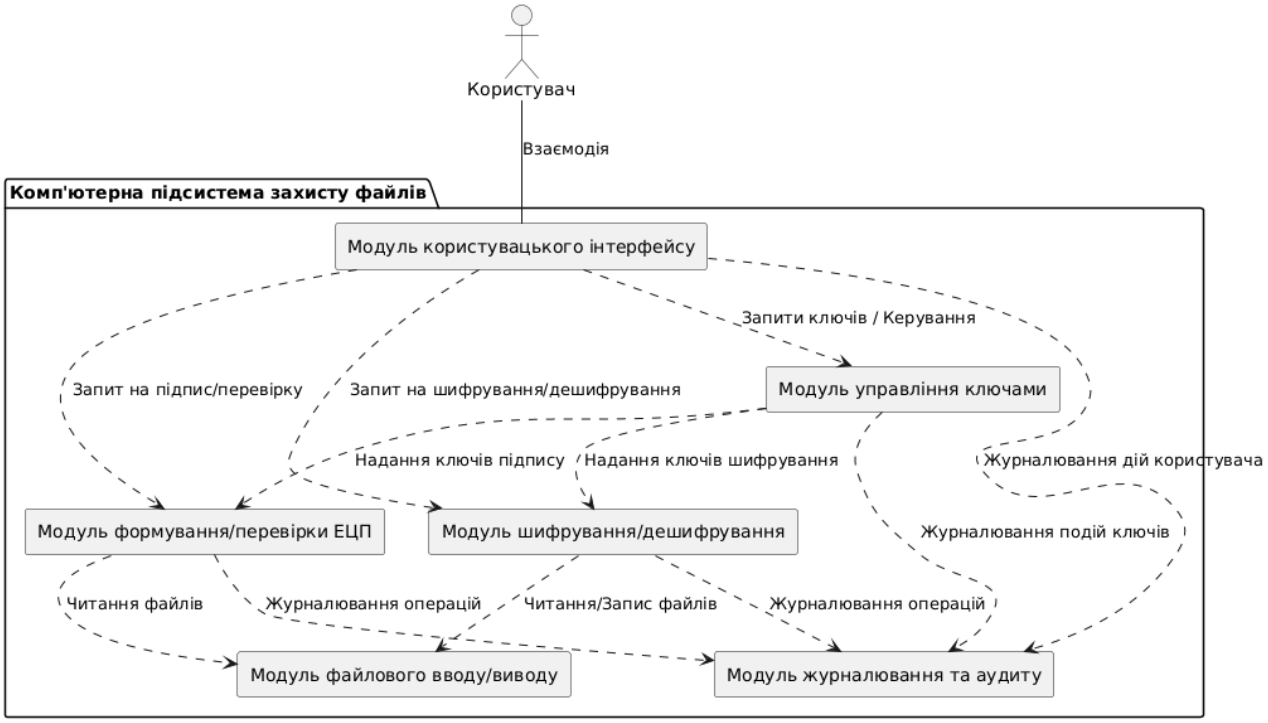

Рисунок 2.1 – Архітектура комп'ютерної підсистеми

Модуль користувацького інтерфейсу (User Interface Module). Це єдина точка взаємодії користувача з підсистемою. Він надає інтуїтивно зрозумілий графічний інтерфейс (GUI) або, за потреби, інтерфейс командного рядка (CLI) для ініціації всіх операцій. Користувач може обирати файли для шифрування, дешифрування, підписання або перевірки підпису, керувати ключами та отримувати повідомлення про статус та результати операцій. Модуль взаємодіє з іншими компонентами, передаючи їм команди та дані, а також відображаючи інформацію, отриману від них.

Модуль управління ключами (Key Management Module): Цей модуль є серцем безпеки підсистеми. Він відповідає за:

– генерацію криптографічних ключів. Безпечно створює пари асиметричних ключів (відкритий та закритий) для користувача. Може також генерувати симетричні ключі, які використовуються для шифрування даних;

– зберігання ключів. Забезпечує надійне зберігання згенерованих ключів. Закриті ключі повинні зберігатися в зашифрованому вигляді або у захищеному сховищі (наприклад, файлове сховище, захищене паролем, або інтеграція з апаратним модулем безпеки – HSM);

– завантаження/Експорт ключів. Дозволяє безпечно завантажувати існуючі ключі або експортувати відкриті ключі для обміну з іншими користувачами.

Модуль шифрування та дешифрування (Encryption/Decryption Module): Цей модуль відповідає за власне криптографічні перетворення даних. Бере вихідний файл і, використовуючи обраний алгоритм симетричного шифрування (наприклад, AES), шифрує його вміст. Симетричний ключ для шифрування даних, у свою чергу, шифрується відкритим ключем одержувача (асиметричне шифрування) і передається разом із зашифрованим файлом. Отримує зашифрований файл та зашифрований симетричний ключ. Дешифрує симетричний ключ закритим ключем отримувача, а потім використовує його для дешифрування вмісту файлу, відновлюючи оригінальні дані.

Модуль формування та перевірки електронного підпису (Digital Signature Module): Цей модуль реалізує функціонал електронного підпису. Обчислює криптографічну хеш-функцію (наприклад, SHA-256) від вмісту файлу. Отриманий хеш підписується закритим ключем відправника. Результатом є електронний підпис, який додається до файлу або передається окремо. Отримує файл, його електронний підпис та відкритий ключ відправника. Обчислює хеш від файлу та порівнює його з хешем, отриманим після дешифрування підпису відкритим ключем. Невідповідність означає, що файл був змінений або підпис підроблений.

Модуль файлового вводу/виводу (File I/O Module): Цей допоміжний модуль забезпечує взаємодію підсистеми з файловою системою. Він відповідає за:

– читання файлів. Безпечно зчитує вміст вихідних файлів;

– запис файлів. Зберігає оброблені файли (зашифровані, дешифровані) та підписи;

– обробка потоків. Забезпечує ефективну потокову обробку даних для великих файлів, мінімізуючи використання пам'яті.

Модуль журналювання та аудиту (Logging & Audit Module): Відповідає за фіксацію всіх значущих подій та операцій, що відбуваються в підсистемі. Це включає:

– запис подій. Генерація ключів, спроби доступу (успішні/невдалі),

операції шифрування/дешифрування, формування/перевірка підписів, виявлені помилки або інциденти безпеки;

– зберігання журналів. Надійне зберігання логів для подальшого аналізу, моніторингу та розслідування інцидентів.

2.3 Програмування комп'ютерної підсистеми перевірки цілісності та достовірності файлів при передачі даних

2.3.1 Реалізація генерації ключів

Сучасні програмні платформи значно спрощують інтеграцію криптографічних функцій, дозволяючи розробникам зосередитися на логіці програми, а не на низькорівневих математичних деталях. Програмісту не потрібно заглиблюватися в складні математичні формули та тонкощі їх реалізації. Натомість можна використовувати вже готові, перевірені алгоритми та методи криптографії, які надаються спеціальними програмними модулями, відомими як провайдери. У випадку з мовою програмування Java, усі необхідні криптографічні засоби вже інтегровані в стандартну бібліотеку Java Cryptography Architecture (JCA) та поставляються провайдером за замовчуванням, який традиційно має назву "SUN" (хоча з розвитком OpenJDK його можуть замінювати інші реалізації, такі як OpenJDK Provider). Таким чином, для використання криптографічних функцій у Java, програмісту достатньо вказати лише бажаний алгоритм та провайдера.

Першим і часто найважливішим кроком у багатьох криптографічних операціях, таких як цифровий підпис або асиметричне шифрування, є генерація пари ключів — відкритого (public key) та закритого (private key). Для цього в Java використовується клас KeyPairGenerator. Щоб отримати конкретний об'єкт цього класу, необхідно викликати його статичний фабричний метод getInstance(). Як аргументи цьому методу передаються рядки, що вказують на конкретний криптографічний алгоритм та бажаного провайдера. Наприклад, для генерації ключів за алгоритмом DSA (Digital Signature Algorithm) з використанням провайдера "SUN" (як показано в Додатку В), код представлено на рис.2.2.

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA", "SUN");
```

Рисунок 2.2 – Фрагмент алгоритму DSA з використанням провайдера "SUN"

Оскільки більшість сучасних криптографічних алгоритмів (особливо ті, що стосуються генерації ключів) є імовірнісними за своєю природою, критично важливо використовувати високоякісне джерело випадковості. Для цього в Java застосовується клас SecureRandom. Цей клас надає криптографічно стійкі псевдовипадкові числа, які необхідні для забезпечення безпеки згенерованих ключів. Об'єкт SecureRandom також отримується за допомогою статичного фабричного методу getInstance(), де можна вказати алгоритм генерації псевдовипадкових чисел (наприклад, "SHA1PRNG") та провайдера як показано на рис.2.3.

```
SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
```

Рисунок 2.2 – Фрагмент алгоритму генерації псевдовипадкових чисел

Після створення джерела випадковості, необхідно проініціалізувати об'єкт keyGen. Це робиться за допомогою методу initialize(), який приймає два параметри: довжину ключа в бітах (чим більша довжина, тим складніше ключ зламати, але тим повільнішими стають операції) та щойно створений об'єкт джерела випадковості як показано на рис.2.3

```
keyGen.initialize(1024, random); // 1024 біти - типова довжина для DSA
```

Рисунок 2.3 – Фрагмент створення об'єкту джерела випадковості

Останнім етапом є безпосереднє виконання генерації пари ключів за допомогою методу generateKeyPair(), який повертає об'єкт типу KeyPair. З цього об'єкта потім можна легко виділити окремо приватний та публічний ключі за допомогою методів getPrivate() та getPublic() відповідно як показано на рис.2.4.

```
KeyPair pair = keyGen.generateKeyPair();
PrivateKey privKey = pair.getPrivate();
PublicKey pubKey = pair.getPublic();
```

Рисунок 2.4 – Фрагмент створення методу generateKeyPair()

Згенеровані privKey (закритий ключ) та pubKey (відкритий ключ) тепер готові до використання в подальших криптографічних операціях, таких як шифрування даних для одержувача (використовуючи його публічний ключ) або підписання документів (використовуючи власний приватний ключ). Важливо пам'ятати, що закритий ключ повинен залишатися суворо конфіденційним, тоді як відкритий ключ може вільно розповсюджуватися.

Цей спрощений підхід, реалізований у Java, демонструє, як потужні криптографічні інструменти стають доступними для широкого кола розробників, дозволяючи їм будувати безпечні застосунки без глибокого занурення у математичні основи.

2.3.3 Створення підпису

Електронний підпис є ключовим компонентом для забезпечення цілісності та автентичності даних. У контексті Java Cryptography Architecture (JCA), процес створення електронного підпису реалізується за допомогою класу Signature. Розглянемо детальніше етапи цього процесу.

На першому етапі необхідно створити екземпляр класу Signature, який буде відповідати за операції підпису. Для цього викликається статичний фабричний метод getInstance(), вказавши бажаний алгоритм підпису та провайдера. Вибір алгоритму є критично важливим: у даному випадку використовується DSA (Digital Signature Algorithm) у поєднанні з SHA-1 (Secure Hash Algorithm 1) як хеш-функцією. Це означає, що спершу від даних буде обчислений хеш SHA-1, а потім цей хеш буде підписаний за допомогою алгоритму DSA. Провайдером, як правило, виступає стандартний провайдер "SUN" (рис.2.5).

```
Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");
```

Рисунок 2.5 – Фрагмент створення екземпляру класу Signature

Після успішного створення об'єкта Signature необхідно ініціалізувати його для операції підписання. На цьому етапі використовується закритий ключ (private key), який був згенерований раніше. Закритий ключ є секретним і відомий лише власнику, що забезпечує унікальність та автентичність створюваного підпису. Метод initSign() класу Signature приймає об'єкт PrivateKey як параметр (Додаток А).

Наступним кроком є передача даних, які необхідно підписати, до об'єкта Signature. Це здійснюється за допомогою методу update(). Цей метод приймає байтовий масив, що представляє дані (або частину даних), які повинні бути включені в процес обчислення хешу та подальшого підписання. Можна викликати update() кілька разів для послідовного додавання даних, якщо файл великий або дані надходять потоком.

Нарешті, на останньому етапі генерується власне електронний підпис. Це відбувається шляхом виклику методу sign() класу Signature. Цей метод завершує процес хешування даних та криптографічно підписує обчислений хеш за допомогою закритого ключа, який був наданий на етапі ініціалізації. Результатом є байтовий масив, що представляє собою сформований електронний підпис (рис.2.6).

```
dsa.initSign(privKey);

dsa.update(dataToSign);

byte[] realSig = dsa.sign();
```

Рисунок 2.5 – Фрагмент генерації електронного підпису

Отриманий realSig (байтовий масив, що є підписом) тепер може бути переданий разом із вихідними даними (або окремо) одержувачу. Одержувач, маючи вихідні дані, підпис та відкритий ключ відправника, зможе перевірити цілісність даних та автентичність їх джерела.

На рис.2.6 представлена структурна схема відображає ключові етапи та

взаємодію компонентів у процесі створення електронного підпису з використанням алгоритму Digital Signature Algorithm (DSA). Ця схема є фундаментальною для розуміння того, як забезпечується автентичність та цілісність даних у криптографічних системах.

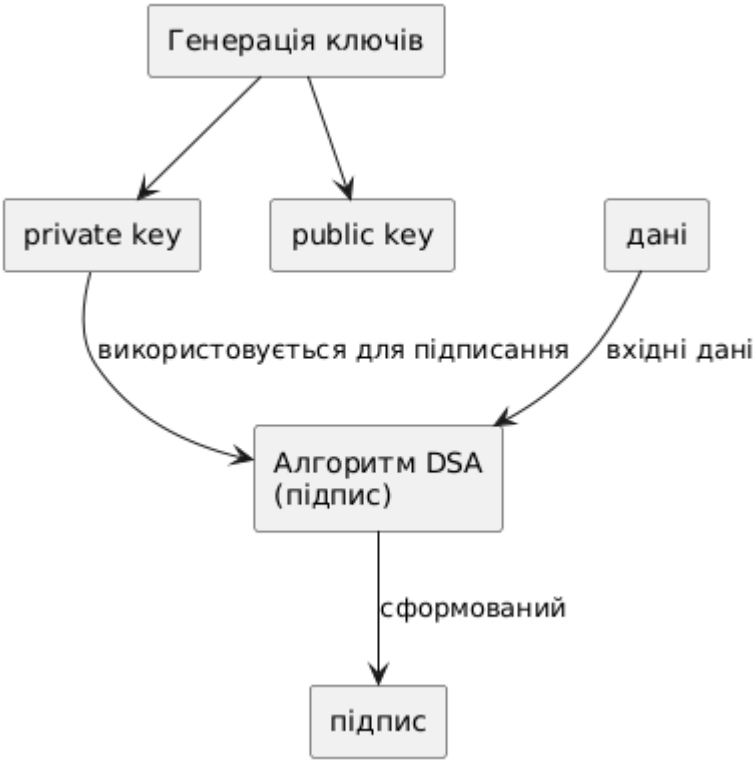

Рисунок 2.6 – Схема генерації підпису

Процес починається з блоку "Генерація ключів". Цей початковий етап символізує створення криптографічно пов'язаної пари ключів, яка є основою асиметричної криптографії. Цей процес, який виконується лише один раз для кожного користувача або сутності, що має намір підписувати дані, призводить до появи двох ключових сутностей: "private key" (закритий ключ) та "public key" (відкритий ключ). Закритий ключ є суто конфіденційним компонентом, унікальним для власника, і зберігається в таємниці, призначений виключно для формування електронного підпису. Відкритий ключ, натомість, є публічним і може вільно розповсюджуватися, призначений для перевірки підписів.

Далі, "private key" (закритий ключ), який є прямим результатом процесу генерації, подається на вхід центрального блоку схеми – "Алгоритм DSA (підпис)".

Це підкреслює його критичну роль: без закритого ключа сформувати дійсний електронний підпис неможливо. Важливо зауважити, що на цій схемі "public key" (відкритий ключ), хоча і генерується разом із закритим, не має прямої стрілки до "Алгоритм DSA (підпис)". Це пояснюється тим, що відкритий ключ не використовується для створення підпису; його функція полягає в перевірці підпису, що є окремим процесом, який не є предметом даної схеми.

Одночасно з закритим ключем, до блоку "Алгоритм DSA (підпис)" надходять безпосередньо "дані" – вихідна інформація (файл, повідомлення, документ тощо), яку необхідно підписати. Це можуть бути будь-які цифрові дані. "Алгоритм DSA (підпис)" є обчислювальною функцією, що виконує дві основні дії: по-перше, вона обчислює криптографічну хеш-функцію (наприклад, SHA-1) від вхідних "даних", створюючи їхній унікальний "відбиток" фіксованої довжини. По-друге, обчислений хеш криптографічно шифрується за допомогою наданого "private key" – це і є власне операція цифрового підпису.

Кінцевим результатом роботи "Алгоритму DSA (підпис)" є блок "підпис". Це згенерований електронний підпис, що являє собою набір криптографічних даних – не сам хеш даних, а їхній хеш, криптографічно оброблений закритим ключем. Цей "підпис" є унікальним для конкретних даних та використаного закритого ключа і може бути переданий разом з оригінальними даними одержувачу. Останній, використовуючи відкритий ключ відправника та окремий процес верифікації, зможе перевірити, чи не були дані змінені після підписання (гарантуючи цілісність) і чи справді вони походять від заявленого власника цього відкритого ключа (забезпечуючи автентичність) [4].

2.3.4 Збереження підпису

Після успішної генерації електронного підпису та отримання необхідних ключів, наступним критично важливим етапом є безпечна передача цих компонентів адресату, а потім – їх верифікація для підтвердження цілісності та автентичності файлів. Важливо розуміти, що закритий ключ (private key) ніколи не посилається

одержувачу; він завжди залишається у відправника, гарантуючи його конфіденційність та унікальність підпису. Натомість, для верифікації підпису одержувачу необхідний відкритий ключ (public key) відправника, який, як випливає з його назви, є публічним і може бути безпечно переданий.

Отже, для успішної верифікації електронного підпису, відправник надсилає одержувачу, наприклад, електронною поштою, через пряме мережеве з'єднання або будь-яким іншим каналом передачі даних, наступні три ключові компоненти.

Відкритий ключ (public key) є публічною частиною пари ключів відправника. Він використовується одержувачем для криптографічного перетворення електронного підпису з метою отримання хешу, який потім буде порівнюватися з хешем від отриманих даних.

Електронний підпис (digital signature) – це безпосередньо згенерований підпис, який є результатом обробки хешу вихідних даних закритим ключем відправника. Він є доказом того, що дані були підписані саме власником закритого ключа.

Вихідні дані (підписаний документ або код) – це оригінальний файл, повідомлення або будь-які інші дані, які були підписані. Саме ці дані отримувач буде перевіряти на цілісність та автентичність.

Отримавши ці три компоненти, одержувач виконує процедуру верифікації, щоб переконатися, що дані не були змінені (цілісність) і що вони справді походять від заявленого відправника (автентичність). Цей процес реалізується також за допомогою алгоритму DSA.

На стороні одержувача виконуються наступні кроки:

– ініціалізація об'єкта Signature для верифікації. Одержувач створює об'єкт Signature, вказуючи той самий алгоритм, що використовувався для підписання (наприклад, "SHA1withDSA"), і ініціалізує його відкритим ключем відправника за допомогою методу initVerify(publicKey);

– додавання отриманих даних. Отримані вихідні дані подаються до об'єкта Signature за допомогою методу update(). Алгоритм внутрішньо обчислює хеш-функцію від цих даних;

– верифікація підпису. Нарешті, одержувач викликає метод verify() об'єкта

Signature, передаючи йому отриманий електронний підпис. Цей метод виконує два ключові кроки: використовує відкритий ключ відправника для дешифрування отриманого електронного підпису, відновлюючи хеш, який був підписаний відправником. Порівнює відновлений хеш з хешем, обчисленим від отриманих вихідних даних.

Результат верифікації. Якщо обидва хеші збігаються, метод verify() повертає true, що означає успішну верифікацію: дані є цілісними, і підпис справжній. Якщо хеші не збігаються, метод повертає false, вказуючи на те, що дані були змінені або підпис недійсний/підроблений.

Представлена структурна схема на рис.2.7 відображає ключові етапи та взаємодію компонентів у процесі верифікації (перевірки) електронного підпису за допомогою алгоритму Digital Signature Algorithm (DSA). Ця схема є критично важливою для підтвердження цілісності отриманих даних та автентичності їхнього відправника [5].

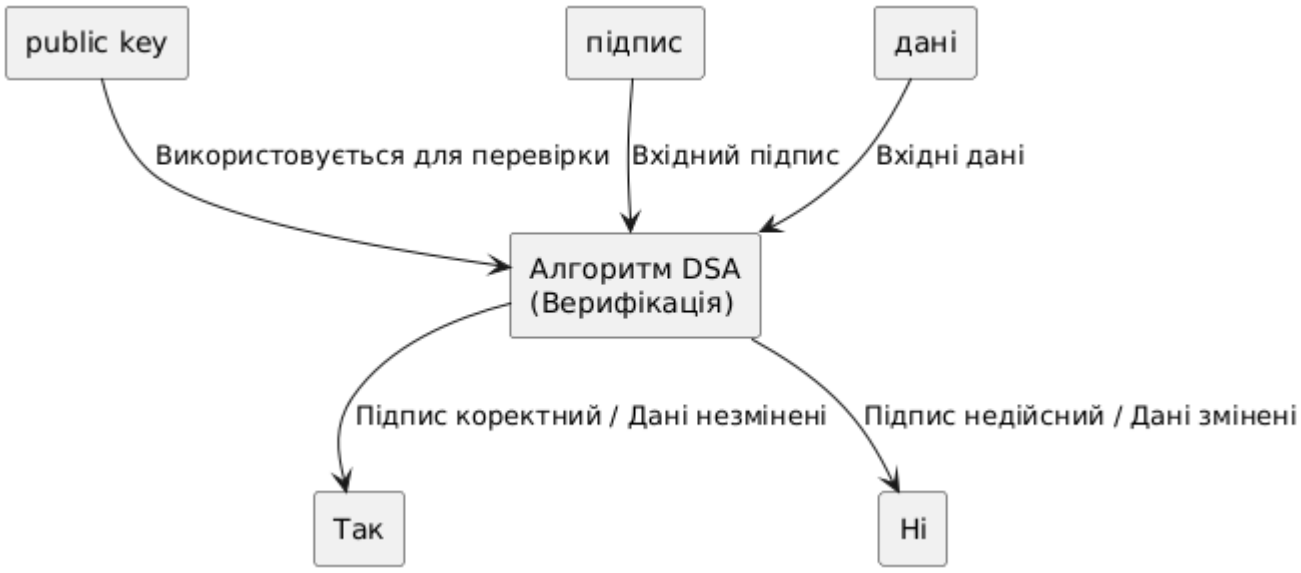

Рисунок 2.7 – Схема верифікації підпису

Процес верифікації починається з трьох ключових вхідних компонентів. Першим є "public key" (відкритий ключ). Цей ключ, згенерований разом із закритим ключем відправника, є публічним і передається одержувачу. Він є абсолютно необхідним для криптографічної перевірки підпису, оскільки використовується для

зворотного перетворення (дешифрування) хешу, який був підписаний закритим ключем відправника. Без відповідного відкритого ключа неможливо перевірити справжність підпису.

Другим вхідним компонентом є власне "підпис" (електронний підпис). Це криптографічна сутність, що була згенерована відправником шляхом хешування вихідних даних та їх підписання власним закритим ключем. Цей підпис є доказом того, що дані були оброблені конкретним закритим ключем. Третім вхідним елементом є "дані" – оригінальний файл, повідомлення або будь-яка інша інформація, яку одержувач отримав і бажає перевірити. Важливо, щоб ці "дані" були саме тими, які були підписані відправником.

Всі три вхідні компоненти – "public key", "підпис" та "дані" – подаються на центральний блок схеми, який називається "Алгоритм DSA (Верифікація)". Цей блок представляє собою обчислювальну функцію, яка виконує складний процес перевірки. Всередині цього алгоритму відбувається декілька ключових операцій. По-перше, алгоритм використовує наданий "public key" для криптографічного дешифрування отриманого "підпису". В результаті цього дешифрування отримується хеш, який був обчислений та підписаний відправником на етапі генерації підпису. По-друге, "Алгоритм DSA (Верифікація)" самостійно обчислює криптографічну хеш-функцію від отриманих "даних".

Завершальним етапом роботи "Алгоритму DSA (Верифікація)" є порівняння двох хешів: хешу, відновленого з підпису за допомогою відкритого ключа, та хешу, обчисленого від отриманих даних. Результат цього порівняння визначає кінцевий вихід схеми. Якщо обидва хеші ідентичні, це свідчить про успішну верифікацію. У цьому випадку алгоритм видає результат "Так", що означає, що підпис є коректним, і дані не були змінені після їх підписання відправником. Якщо ж хеші не збігаються, це означає, що або підпис є недійсним (наприклад, підробленим, або підписаний іншим ключем), або отримані дані були змінені після підписання. У цьому випадку алгоритм видає результат "Ні", сигналізуючи про недійсність підпису або компрометацію даних. Таким чином, ця схема демонструє, як асиметрична криптографія дозволяє незалежно перевіряти цілісність та автентичність інформації

без необхідності ділитися секретним закритим ключем.

2.3.5 Робота з ключем. Зчитування з файлу і перетворення в PrivateKey

Одержувач, як правило, отримує відкритий ключ відправника у вигляді послідовності байтів (наприклад, у файлі або як частину мережевого протоколу). Першим кроком є перетворення цього байтового масиву (наприклад, byte[] encKey) в об'єкт класу PublicKey, який є необхідним для криптографічних операцій у Java.

Для цього використовується клас KeyFactory, який здатний відновити об'єкт класу Key (батьківського класу для PrivateKey та PublicKey) на основі так званої специфікації ключа. Специфікація ключа залежить від стандарту, за яким був згенерований ключ. У випадку, коли ключ був згенерований за допомогою провайдера "SUN" (що є типовим для стандартних реалізацій Java), він відповідає стандарту X.509.

Спочатку байтовий масив відкритого ключа (encKey) інкапсулюється в об'єкт X509EncodedKeySpec. Цей клас є специфічним для стандартів, що використовують кодування X.509, що є поширеним для публічних ключів.

Потім створюється екземпляр класу KeyFactory, який відповідає конкретному криптографічному алгоритму (у цьому випадку "DSA") та провайдеру ("SUN").

Нарешті, за допомогою методу generatePublic() класу KeyFactory з об'єкта pubKeySpec відновлюється повноцінний об'єкт PublicKey, готовий до використання.

Таким чином, отриманий pubKey тепер представляє відкритий ключ відправника у форматі, придатном для використання в Java.

Отриманий електронний підпис також має бути перетворений у байтовий масив (наприклад, byte[] sigToVerify), якщо він ще не є таким (наприклад, якщо він був зчитаний з файлу). Після цього, необхідно створити об'єкт класу Signature, аналогічно тому, як це робилося на стороні відправника для формування підпису.

Потім об'єкт sig ініціалізується для режиму верифікації за допомогою отриманого PublicKey відправника.

На цьому етапі об'єкт Signature готовий приймати дані для обчислення хешу та

подальшого порівняння. Отримані дані, цілісність яких необхідно перевірити, подаються до об'єкта sig за допомогою методу update(). Цей процес аналогічний тому, як дані подавалися під час створення підпису. Метод update() може бути викликаний кілька разів, якщо дані надходять частинами або файл є великим.

Завершальним етапом роботи одержувача є, власне, отримання відповіді на питання про правдивість підпису та цілісність даних. Це досягається за допомогою методу verify() об'єкта класу Signature. Цей метод приймає як параметр байтовий масив отриманого підпису (sigToVerify) і виконує всі необхідні криптографічні перевірки.

Метод verify() виконує такі ключові операції:

– дешифрує отриманий sigToVerify за допомогою pubKey, щоб отримати оригінальний хеш, який був підписаний відправником;

– обчислює хеш від даних, які були передані методом update() на попередньому етапі;

– порівнює два хеші.

Результатом методу verify() є булеве значення (boolean verifies). Значення true означає, що наданий підпис (sigToVerify) є дійсно коректним підписом для отриманих даних (receivedData), і він був створений за допомогою відповідного закритого ключа, що відповідає наданому відкритому ключу (pubKey). Це підтверджує цілісність даних (вони не були змінені) та їхню автентичність (вони походять від заявленого відправника). Значення false, натомість, свідчить про те, що підпис недійсний – дані могли бути змінені, або підпис підроблений, або використано невідповідний ключ.

Слід зазначити, що перший етап, пов'язаний з роботою відправника (генерація ключів та формування підпису), вимагає певного часу. Наприклад, для комп'ютера класу Intel Pentium III з частотою 733 МГц час генерації ключів становить приблизно 10 секунд. Проте, час, що витрачається на верифікацію підпису, є на порядок меншим, що робить цей процес дуже ефективним для масової перевірки даних.

Крім розглянутого алгоритму DSA, Java Cryptography Architecture надає багато

інших засобів криптографічного захисту, таких як RSA, DES (Data Encryption Standard), AES (Advanced Encryption Standard) та інші. Їх використання подібне до представленого DSA, вимагаючи лише зміни назви алгоритму в методі getInstance() та, можливо, використання відповідних специфікацій ключів. Це забезпечує високу гнучкість для розробників у виборі оптимальних криптографічних рішень для конкретних задач [10,11].

2.3.6 Реалізація алгоритму DES для шифрування і розшифровки файлу

Симетричне шифрування є одним з фундаментальних методів криптографічного захисту інформації, при якому для шифрування та дешифрування даних використовується один і той же криптографічний ключ. Цей метод відомий своєю високою швидкістю та ефективністю, що робить його ідеальним для шифрування великих обсягів даних. Розглянемо детальніше реалізацію симетричного шифрування в Java на прикладі алгоритму DES (Data Encryption Standard).

У Java, основна функціональність для роботи з криптографічними алгоритмами шифрування реалізована в класі javax.crypto.Cipher. Цей клас є універсальним інтерфейсом для доступу до різноманітних криптографічних примітивів [12].

Для початку роботи необхідно створити екземпляр класу Cipher за допомогою його статичного фабричного методу getInstance(). Цей метод приймає як параметр рядок, що вказує на назву криптографічного алгоритму шифрування, а також, за бажанням, режим роботи та схему доповнення (padding). У нашому випадку, для використання базового алгоритму DES.

Наступним кроком є ініціалізація екземпляра класу Cipher та вказівка режиму його роботи:

- для шифрування: chr.init(Cipher.ENCRYPT_MODE, key);
- для дешифрування: chr.init(Cipher.DECRYPT_MODE, key).

Як видно, у методі init() з'явився новий важливий параметр: key. Це 56-бітний

ключ алгоритму DES, представлений об'єктом типу javax.crypto.SecretKey. Цей секретний ключ є центральним елементом симетричного шифрування. Об'єкт SecretKey може бути згенерований за допомогою класу javax.crypto.KeyGenerator.

Безпосереднє шифрування або дешифрування виконується методом doFinal() класу Cipher. Ця функція приймає на вхід байтовий масив вихідних даних (відкритого тексту для шифрування або шифротексту для дешифрування) і повертає також байтовий масив, але вже відповідно перетворених даних. Те, що саме (шифрування чи дешифрування) виконуватиме функція doFinal(), залежить від режиму, який було вказано в першому параметрі функції init().

Для забезпечення можливості збереження та подальшого використання згенерованих криптографічних ключів (особливо SecretKey у симетричному шифруванні) необхідно реалізувати механізм їхнього персистентного зберігання. Одним із поширених способів у Java є серіалізація.

Серіалізація — це процес перетворення об'єкта в потік байтів, що дозволяє зберігати його в файлі, передавати через мережу або відновлювати пізніше. Для того, щоб об'єкт був серіалізованим, клас цього об'єкта повинен реалізовувати інтерфейс java.io.Serializable.

Інтерфейс java.io.Serializable є так званим інтерфейсом-маркером (marker interface), оскільки він не містить жодних методів. Його єдина мета полягає в тому, щоб слугувати сигналом для механізму серіалізації Java (Java Serialization API) про те, що об'єкт, який реалізує даний інтерфейс, може бути серіалізованим. Класи SecretKey (та їхні конкретні реалізації), як правило, вже реалізують Serializable або є частиною ієрархії, що підтримує серіалізацію, тому часто достатньо просто працювати з ними як з серіалізованими об'єктами [13, 14].

Отже, якщо ми маємо об'єкт класу SecretKey, який реалізує інтерфейс java.io.Serializable, наступним кроком стане написання алгоритму, що виконує серіалізацію екземпляра класу SecretKey до файлу (або іншого потоку). Це дозволить зберегти згенерований ключ і завантажити його згодом для використання в операціях дешифрування або повторного шифрування тих самих даних.

```
import java.io.*;
import javax.crypto.SecretKey;
import javax.crypto.KeyGenerator;

public class KeyPersistence {

    public static void saveKey(SecretKey key, String fileName) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fileName))) {
            oos.writeObject(key);
            System.out.println("Ключ успішно збережено у файл: " + fileName);
        }
    }

    public static SecretKey loadKey(String fileName) throws IOException, ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileName))) {
            SecretKey loadedKey = (SecretKey) ois.readObject();
            System.out.println("Ключ успішно завантажено з файлу: " + fileName);
            return loadedKey;
        }
    }

    public static void main(String[] args) throws Exception {
        // 1. Генерація DES ключа
        KeyGenerator keyGen = KeyGenerator.getInstance("DES");
        SecretKey originalKey = keyGen.generateKey();
        System.out.println("Згенеровано оригінальний ключ: " + originalKey.getAlgorithm());

        String keyFileName = "des_secret.key";

        // 2. Серіалізація ключа
        saveKey(originalKey, keyFileName);

        // 3. Десеріалізація ключа
        SecretKey loadedKey = loadKey(keyFileName);
        System.out.println("Завантажено ключ: " + loadedKey.getAlgorithm());

        // Перевірка, що ключі ідентичні (можна порівняти їх закодовані форми)
        if (java.util.Arrays.equals(originalKey.getEncoded(), loadedKey.getEncoded())) {
            System.out.println("Оригінальний та завантажений ключі ідентичні.");
        } else {
            System.out.println("Помилка: ключі не ідентичні!");
        }

        // Тепер loadedKey можна використовувати для шифрування/дешифрування
        // Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        // cipher.init(Cipher.ENCRYPT_MODE, loadedKey);
        // byte[] encrypted = cipher.doFinal("Hello, World!".getBytes());
        // ...
    }
}
```

Рисунок 2.8 – Приклад серіалізації SecretKey до файлу

2.4 Розробка інтерфейсу програми

Розробка зручного та інтуїтивно зрозумілого інтерфейсу користувача є ключовим етапом у створенні будь-якого програмного забезпечення, особливо для криптографічних підсистем, де помилки користувача можуть мати серйозні наслідки. Інтерфейс має максимально спростити взаємодію з функціями шифрування, дешифрування, підписання та перевірки цифрового підпису, приховуючи складність базових криптографічних операцій [11].

Виходячи з наданих зразків, архітектура інтерфейсу базується на принципах простоти та функціональності. Головне вікно програми має бути мінімалістичним,

містити чотири основні кнопки, що чітко відповідають ключовим криптографічним діям: "Закодувати файл" (шифрування), "Розшифрувати файл" (дешифрування), "Підписати файл" (формування цифрового підпису) та "Перевірити цифровий підпис" (верифікація підпису). Така структура забезпечує легке орієнтування користувача в доступних функціях (рис.2.9).

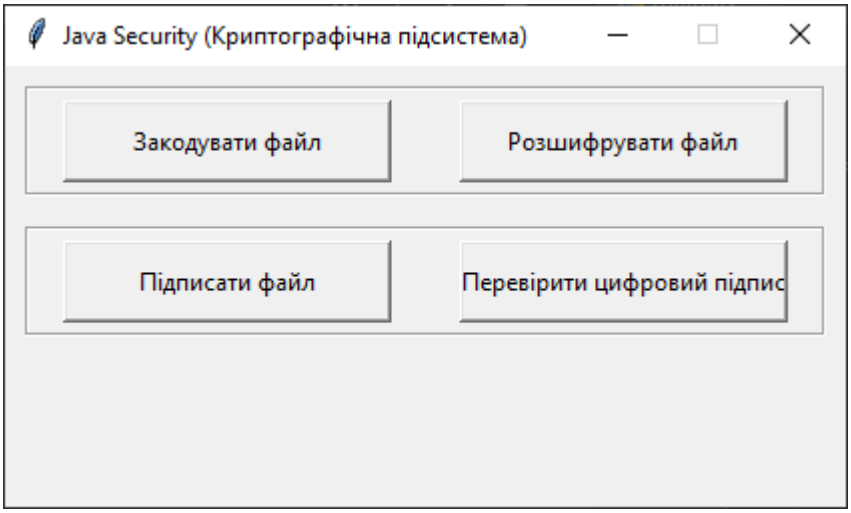

Рисунок 2.9 – Головне вікно програми

Кожна з цих кнопок при активації повинна ініціювати стандартний системний діалог вибору файлу, дозволяючи користувачеві зручно навігувати по файловій системі та вказувати об'єкти для обробки. Після завершення будь-якої криптографічної операції – незалежно від того, чи була вона успішною, чи завершилася помилкою – програма повинна надавати користувачеві чітке, інформативне повідомлення про результат. Це може бути підтвердження успішного виконання, наприклад, "Файл закодовано успішно!", або детальне повідомлення про причину виникнення проблеми, наприклад, "Не вдалося розкодувати файл. Можливо, невірний ключ або пошкоджений файл.", що допомагає користувачеві зрозуміти ситуацію та, за можливості, виправити її.

Крім того, важливо забезпечити надійну обробку винятків. Це включає коректну реакцію на такі ситуації, як відсутність вказаних файлів, використання невірних криптографічних ключів, спроба обробки пошкоджених даних або інші непередбачені обставини. Програма повинна не просто "падати", а коректно реагувати, інформуючи користувача про виниклу проблему. Загальна архітектура

інтерфейсу має бути модульною, дозволяючи легке розширення функціоналу, та крос-платформною, що забезпечує стабільну роботу на різних операційних системах. Це досягається завдяки використанню стандартних, вбудованих бібліотек для розробки графічного інтерфейсу, таких як tkinter у Python, що сприяє швидкій та ефективній побудові надійної та зручної графічної оболонки для криптографічних операцій.

2.5 Інструкція користувача

Для того, щоб розпочати процес шифрування файлу, вам необхідно натиснути на кнопку "Закодувати файл" на головному екрані програми. Ця кнопка є однією з чотирьох основних функцій головного меню, як показано на Рис. 2.9.

Після натискання кнопки " Закодувати файл " програма автоматично відкриє стандартне системне діалогове вікно для вибору файлу. Це вікно дозволить вам навігувати по директоріях вашого комп'ютера. У цьому вікні (рис. 2.10) вам потрібно знайти та вибрати файл, який ви бажаєте зашифрувати. Після вибору файлу натисніть кнопку "Відкрити" у цьому діалоговому вікні.

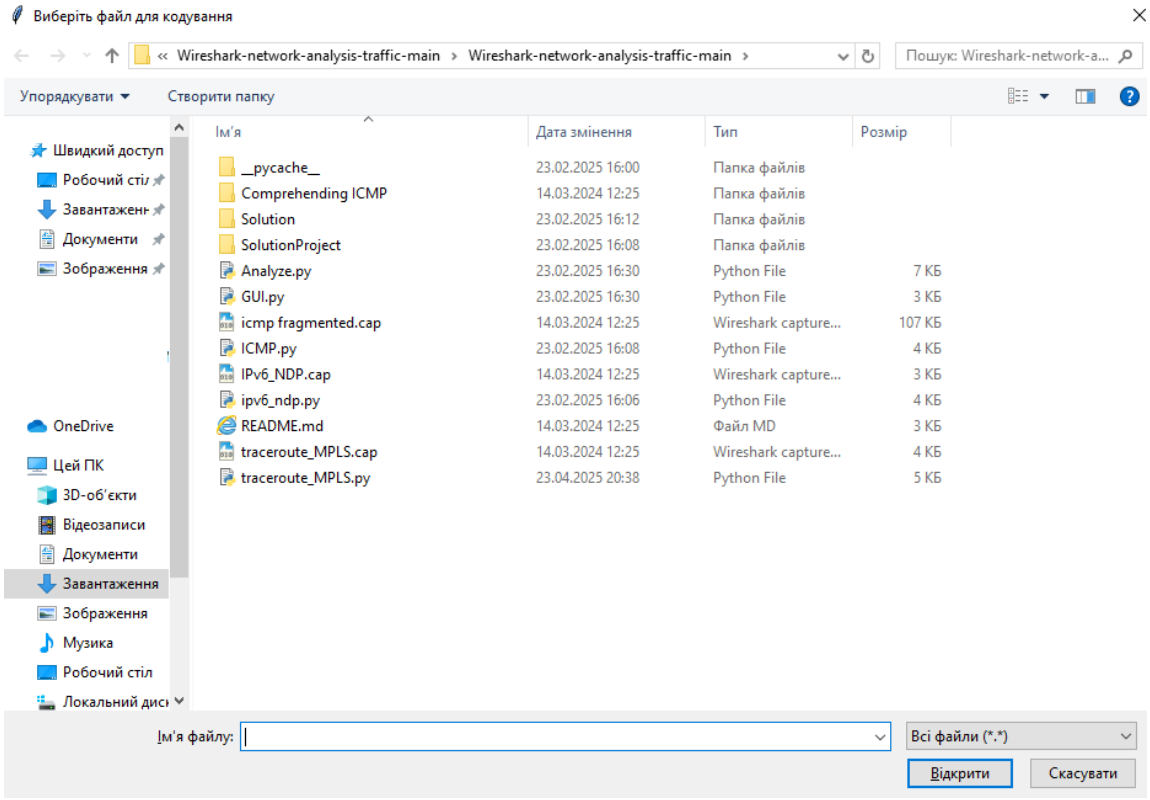

Рисунок 2.10 – Вибір файлу для шифрування

Одразу після вибору вихідного файлу, програма відкриє ще одне стандартне діалогове вікно, але цього разу для збереження файлів. Це вікно призначене для того, щоб ви вказали, куди програма має зберегти зашифровану версію вашого файлу. Вам необхідно:

– вибрати бажану директорію для збереження;

– ввести назву для закодованого файлу. Зазвичай, до назви файлу додається розширення, що вказує на його зашифрований стан (наприклад, .enc, .des або просто залишається оригінальне розширення, якщо це не створює плутанини). Натисніть "Зберегти" у цьому діалоговому вікні.

Після вибору місця збереження зашифрованого файлу, з'явиться ще одне діалогове вікно збереження файлу. Цього разу програма попросить вас вказати місце для збереження файлу ключів. Цей файл міститиме секретний ключ, який є абсолютно необхідним для подальшого розшифрування даних. Збереження цього ключа є надзвичайно важливим! Без нього ви не зможете дешифрувати зашифрований файл.

Оберіть безпечне місце для збереження цього файлу. Рекомендується зберігати ключ окремо від зашифрованого файлу, і, якщо можливо, на захищеному носії.

Вкажіть назву файлу ключа (наприклад, my_secret_key.key). Натисніть "Зберегти" або "ОК".

Після успішного завершення всіх етапів шифрування файлу за алгоритмом DES, програма видасть повідомлення типу MessageBox (спливаюче вікно з інформацією). Це повідомлення підтверджує, що кодування файлу пройшло успішно. Вигляд цього вікна буде подібний до Рис. 2.11, і, ймовірно, міститиме текст на кшталт "Файл закодовано успішно!".

Після отримання цього повідомлення, ви можете бути впевнені, що ваш файл був зашифрований, і його секретний ключ збережений у вказаному місці. Пам'ятайте, що безпека ваших даних безпосередньо залежить від конфіденційності та надійного зберігання цього секретного ключа.

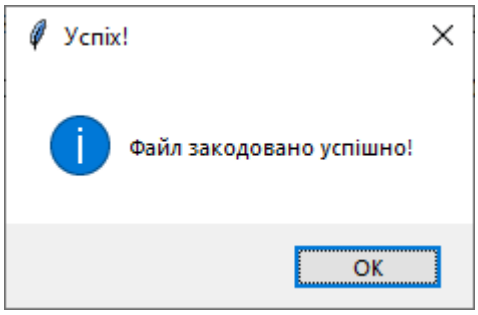

Рисунок 2.10 – Стандартне вікно повідомлення

Для розшифрування раніше закодованого файлу користувач має обрати пункт меню "Розшифрувати файл" на головному вікні програми (Рисунок 2.9). Далі програма послідовно запросить три ключові компоненти: шлях до зашифрованого файлу, який необхідно розшифрувати; адресу для збереження вже декодованого (розшифрованого) файлу; і найголовніше – шлях до файлу з ключем шифрування, без якого дешифрування неможливе. У разі збігу наданого ключа з тим, яким було зашифровано файл, відбудеться успішне розшифрування, і програма повідомить про це. Якщо ж ключ виявиться невідповідним, буде видано повідомлення про неможливість розшифрування через невідповідність файлу ключів (рис.2.11).

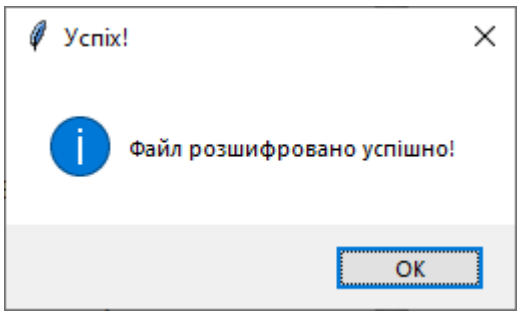

Рисунок 2.11 – Результат "Розшифрувати файл"

Програма також підтримує функціонал роботи з цифровими підписами, що забезпечує автентифікацію та перевірку цілісності даних. Для створення цифрового підпису файлу необхідно вибрати пункт меню "Підписати файл" на головному екрані програми (Рисунок 2.12). Після цього, у стандартному діалоговому вікні відкриття файлу, користувач вказує шлях до файлу, який має бути підписаний. Після успішної операції підписання програма виведе повідомлення про успішне створення цифрового підпису. У вибраній директорії (або у директорії оригінального файлу)

буде створено два нові файли: один з таким самим ім'ям і розширенням .sig, що є власне цифровим підписом, а інший з розширенням .pubkey, який є публічним ключем, необхідним для перевірки цього підпису іншими сторонами.

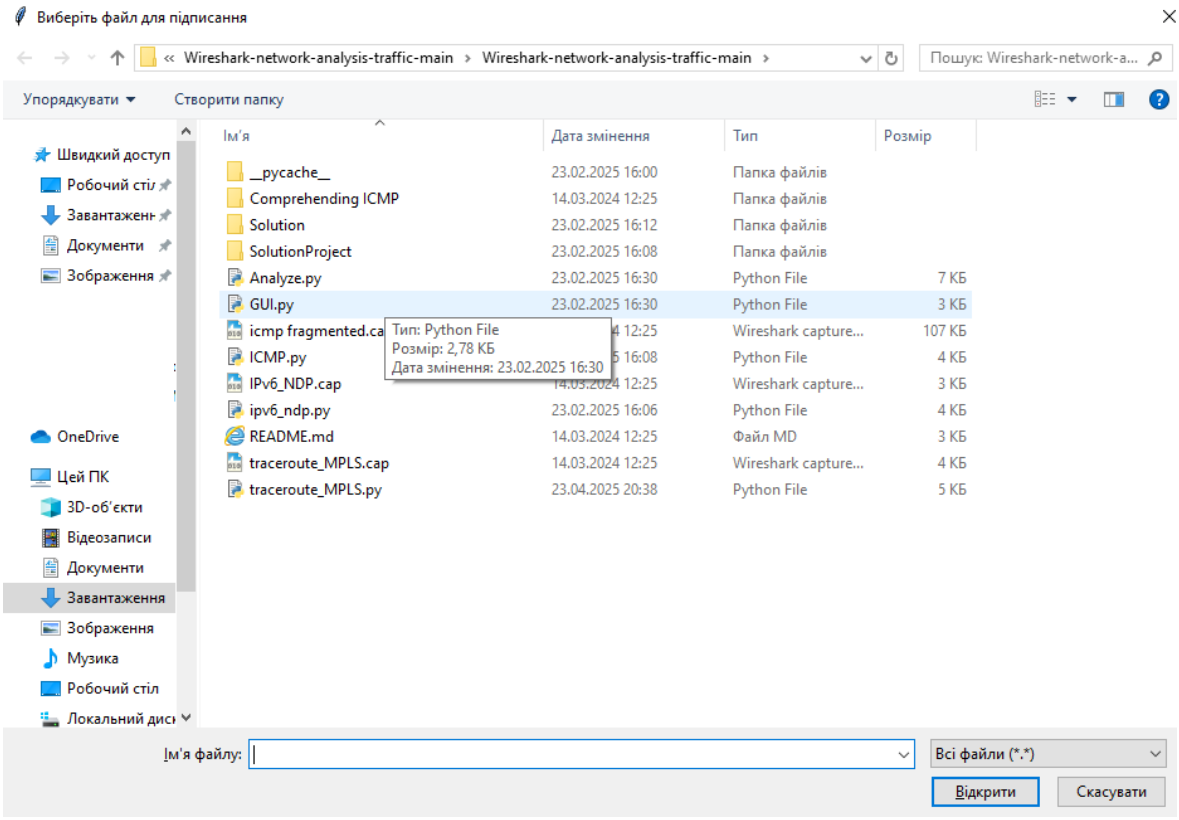

Рисунок 2.12 – Вибір файлу для підписання

Для перевірки цифрового підпису файлу слід обрати пункт меню "Перевірити цифровий підпис" на головному екрані програми (Рисунок 2.13). Ця операція є критично важливою для верифікації автентичності та цілісності даних. Програма послідовно запросить у користувача три компоненти: шлях до файлу, який перевіряється (оригінальний документ); шлях до файлу з цифровим підписом (.sig); та шлях до файлу з публічним ключем відправника (.pubkey). У разі коректного вказання всіх трьох файлів та, що найважливіше, незмінності перевіряємого файлу з моменту його підписання, програма видасть повідомлення про успішну верифікацію. Це підтверджує, що дані не були скомпрометовані, і їхній підпис є дійсним. Натомість, якщо хоча б один біт в одному з цих трьох файлів був модифікований або ключі не співпадають, програма видасть повідомлення про несанкціонований доступ або недійсність підпису, сигналізуючи про можливе

пошкодження або підробку даних.

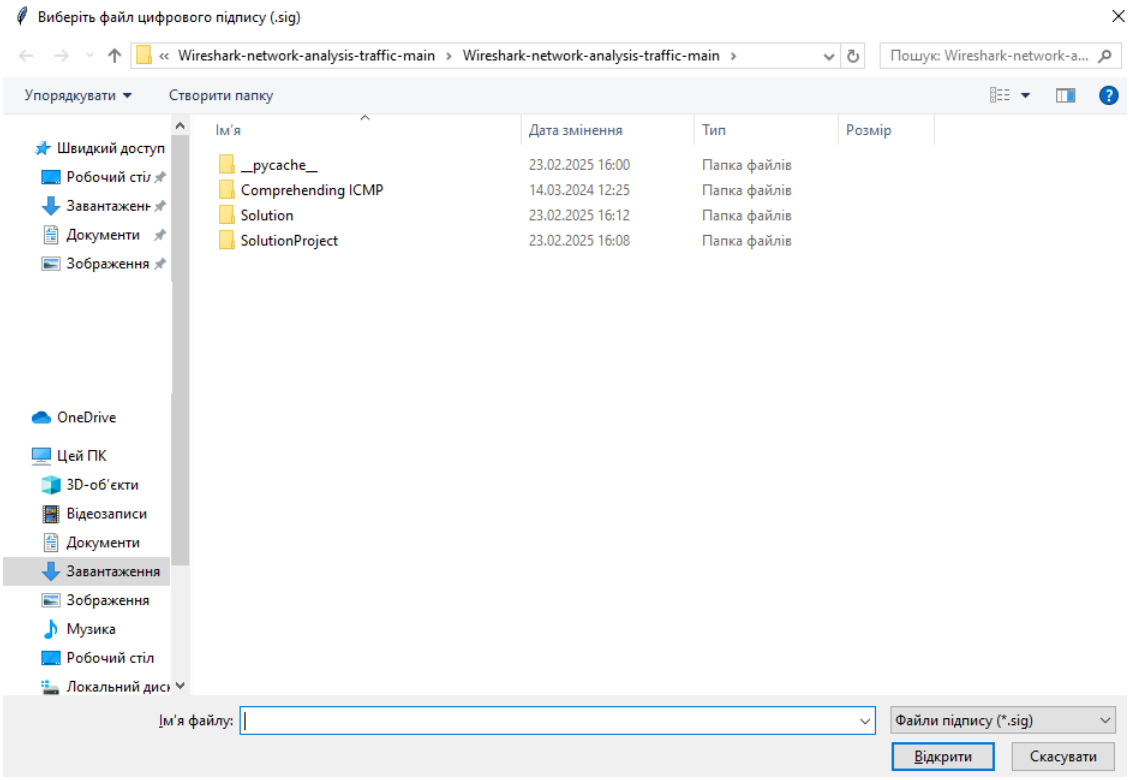

Рисунок 2.13 – Вибір файлу для "Перевірити цифровий підпис"

ВИСНОВКИ

У процесі виконання даної кваліфікаційної роботи було успішно розроблено та реалізовано комп'ютерну підсистему, призначену для забезпечення перевірки цілісності та достовірності файлів під час їх передачі. Досягнуті результати підтверджують ефективність застосованого підходу та відповідність розробленої системи заявленим цілям.

У рамках роботи було проведено глибокий аналіз існуючих криптографічних алгоритмів та методів забезпечення безпеки даних, зокрема цілісності та автентичності. Детально розглянуто принципи симетричного та асиметричного шифрування, а також механізми функціонування цифрового підпису. Це дозволило обґрунтувати вибір алгоритмів DES (для симетричного шифрування) та DSA (для цифрового підпису) як оптимальних для реалізації поставлених завдань, враховуючи їхні характеристики та застосовність у типових сценаріях передачі даних

Обґрунтовано вибір мови програмування Java та її криптографічних розширень (Java Cryptography Architecture, JCA) як основної платформи для реалізації підсистеми. JCA надає надійний та гнучкий фреймворк для роботи з криптографічними примітивами, забезпечуючи високий рівень безпеки та стандартизації. Здатність Java до крос-платформності також є значною перевагою, що дозволяє використовувати підсистему на різних операційних системах без значних модифікацій.

Розроблено та реалізовано криптографічні модулі. Реалізовано функціонал шифрування файлів за допомогою алгоритму DES, що забезпечує конфіденційність даних. Досягнуто ефективної передачі та обробки даних, а також їхнього успішного дешифрування за наявності коректного ключа. Окремо вирішено питання збереження та завантаження секретних ключів за допомогою механізму серіалізації, що є базовим для подальшого використання ключа. Розроблено функціонал генерації унікальної пари асиметричних ключів (закритого та відкритого) та формування електронного підпису файлу за допомогою закритого ключа. Найважливішим досягненням є реалізація механізму верифікації цифрового підпису,

що дозволяє однозначно підтвердити цілісність отриманих даних та їхню автентичність, використовуючи відкритий ключ відправника.

Розроблений графічний інтерфейс програми (GUI), заснований на бібліотеці tkinter (у варіанті для Python, що був продемонстрований), є інтуїтивно зрозумілим та забезпечує легку взаємодію користувача з криптографічними функціями. Він дозволяє зручно обирати файли для обробки, вказувати шляхи збереження результатів, а також отримувати чіткі повідомлення про успішність операцій або виникнення помилок. Це суттєво підвищує зручність використання системи для кінцевих користувачів.

Проведено тестування розроблених модулів, яке підтвердило їхню працездатність та коректність виконання криптографічних операцій. Зокрема, перевірено успішність шифрування та дешифрування файлів, а також безпомилковість створення та верифікації цифрових підписів. Доведено, що підсистема ефективно виявляє будь-які модифікації файлів після їх підписання, що є ключовим показником забезпечення цілісності.

Розроблена комп'ютерна підсистема має значну практичну цінність, оскільки вона може бути використана як самостійний інструмент для захисту даних, так і як модульний компонент у більших програмних комплексах, що потребують функцій перевірки цілісності та достовірності інформації при передачі. Вона може застосовуватися у сферах, де конфіденційність та незмінність даних є пріоритетом, наприклад, в системах документообігу, електронної пошти, файлообмінних сервісах тощо.

ПЕРЕЛІК ПОСИЛАНЬ

1. Основи криптографії: навчальний посібник. / Укл. В.В. Гринюк. – К.: КПІ ім. Ігоря Сікорського, 2017. – 150 с.

2. Криптографія [Електронний ресурс] – Режим доступу до ресурсу: http://uk.wikipedia.org/wiki/Криптографія.

3. Симетричні алгоритми шифрування [Електронний ресурс] – Режим доступу до ресурсу: http://uk.wikipedia.org/wiki/Симетричні_алгоритми_шифрування.

4. RSA [Електронний ресурс] – Режим доступу до ресурсу: http://ru.wikipedia.org/wiki/RSA.

5. DES [Електронний ресурс] – Режим доступу до ресурсу: http://ru.wikipedia.org/wiki/DES.

6. Доля В. І. Криптографічний захист інформації. – Харків: ХНАДУ, 2018. – 230 с.

7. Стандарти криптографічного захисту інформації (ДСТУ, ISO/IEC).

8. Kudinov, V. A., & Mykhailov, S. M. (2020). Security Analysis of Data Transmission Protocols in Distributed Systems. Proceedings of the International Conference on Information Security and Cryptology (ISC), 123-130.

9. Lisovets, V. V., & Bondarenko, I. A. (2019). Investigation of cryptographic transformation methods for ensuring data integrity. Scientific Journal of National Technical University "Kharkiv Polytechnic Institute", (3), 45-50.

10. Melnyk, O. P., & Koval, A. S. (2021). Recent trends in cryptanalysis of symmetric ciphers and countermeasures. Journal of Cyber Security and Data Protection, 7(2), 87-95.

11. PyCryptodome Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://pycryptodome.readthedocs.io/en/latest/.

12. Cryptography.io Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://cryptography.io/en/latest/.

13. Bouncy Castle Crypto APIs [Електронний ресурс] – Режим доступу до ресурсу: https://www.bouncycastle.org/.

14. FIPS PUB 46-3: Data Encryption Standard (DES) [Електронний ресурс] – Режим доступу до ресурсу: https://csrc.nist.gov/pubs/fips/46-3/archive.

ДОДАТОК А

Комп'ютерна підсистема перевірки цілісності та достовірності файлів при передачі даних

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

КОМП'ЮТЕРНА ПІДСИСТЕМА ПЕРЕВІРКИ ЦІЛІСНОСТІ ТА ДОСТОВІРНОСТІ ФАЙЛІВ ПРИ ПЕРЕДАЧІ ДАНИХ

Текст програми

804.02070743.25018-01 18 01

Листів 5

2025

АНОТАЦІЯ

Дана програма представляє собою комп'ютерну підсистему, розроблену для забезпечення критично важливих аспектів безпеки даних: перевірки цілісності та достовірності файлів під час їх передачі. Вона досягає цього завдяки інтеграції та ефективному використанню класичних криптографічних алгоритмів симетричного шифрування (DES) та асиметричного цифрового підпису (DSA).

Програма призначена для надання користувачам надійного інструменту для захисту їхньої інформації. Вона дозволяє шифрувати файли для забезпечення їхньої конфіденційності, гарантуючи, що лише авторизовані особи, які володіють правильним секретним ключем, зможуть отримати доступ до вихідного вмісту. Одночасно, підсистема забезпечує формування та верифікацію цифрових підписів, що є фундаментальним для підтвердження того, що файл не був змінений (цілісність) після його підписання та що він походить від заявленого відправника (достовірність).

Програма розроблена з використанням стандартних бібліотек Java (JCA), що забезпечує її високу надійність, безпеку та крос-платформну сумісність. Вона має зручний, інтуїтивно зрозумілий графічний інтерфейс користувача, який спрощує виконання складних криптографічних операцій, таких як вибір файлів, збереження зашифрованих даних та ключів, а також обробку результатів перевірки. Ця підсистема може бути використана в будь-якій сфері, де виникає потреба у безпечній передачі даних – від особистого обміну конфіденційною інформацією до інтеграції у більш складні корпоративні системи документообігу, електронної пошти чи архівування, де цілісність та достовірність інформації є найвищим пріоритетом.

ЗМІСТ

C.

1. DigitalSignature …….……..……..4

Код класса DigitalSignature

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.Signature;
import java.security.spec.X509EncodedKeySpec;
public class DigitalSignature {
public static void saveToFile (byte[] info, String filename) {
try {
FileOutputStream fos = new FileOutputStream(filename);
fos.write(info);
fos.close();
}
catch (Exception e){}
}
public static byte[] readFromFile (String fileName) {
byte[] info;
try {
FileInputStream fis = new FileInputStream(fileName);
info = new byte[fis.available()];
fis.read(info);
fis.close();
}
catch (Exception e) {info = new byte[0];}
return(info);
}
public static boolean CreateDigitalSignatureForFile(String puth)
{
try
{
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA", "SUN");
SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");
```

```
keyGen.initialize(1024, random);
KeyPair pair = keyGen.generateKeyPair();
PrivateKey priv = pair.getPrivate();
PublicKey pub = pair.getPublic();
Signature dsa = Signature.getInstance("SHA1withDSA", "SUN");
dsa.initSign(priv);
FileInputStream fis = new FileInputStream(puth);
BufferedInputStream bufin = new BufferedInputStream(fis);
byte[] buffer = new byte[1024];
int len;
while (bufin.available() != 0)
{
len = bufin.read(buffer);
dsa.update(buffer, 0, len);
}
bufin.close();
byte[] realSig = dsa.sign();
saveToFile (realSig,puth+".sig");
byte[] key = pub.getEncoded();
saveToFile (key,puth+".pubkey");
//byte[] priv_key = priv.getEncoded();
//saveToFile (priv_key,"privkey_"+puth);
return true;
}
catch (Exception e){}
return false;
}
public static boolean TestedByDigitalSignature(String puth, String sign_puth, String
pubkey_puth){
try{
byte[] encKey = readFromFile(pubkey_puth);
X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(encKey);
KeyFactory keyFactory = KeyFactory.getInstance("DSA", "SUN");
PublicKey pubKey = keyFactory.generatePublic(pubKeySpec);
byte[] sigToVerify = readFromFile(sign_puth);
Signature sig = Signature.getInstance("SHA1withDSA", "SUN");
sig.initVerify(pubKey);
FileInputStream datafis = new FileInputStream(puth);
BufferedInputStream bufin = new BufferedInputStream(datafis);
```

```
byte[] buffer = new byte[1024];
int len;
while (bufin.available() != 0)
{
len = bufin.read(buffer);
sig.update(buffer, 0, len);
}
bufin.close();
boolean verifies = sig.verify(sigToVerify);
return verifies;
}
catch(Exception e){}
return false;
}
}
```