Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК/_____/
підпис
«___» _____ 2024 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»
на тему: “Програмний засіб асиметричного шифрування файлів на основі еліптичних кривих”

Студент групи КН-41 Іван БОРАК _____
(підпис)

Керівник роботи Степан ІВАСЬЄВ _____
(підпис)

Консультанти:

з техніко-економічного
обґрунтування Любов МЕЛЕНЧУК _____
(підпис)

нормоконтролер Надія ГАВРИШКІВ _____
(підпис)

Тернопіль - 2024

Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК/_____/
підпис
«___» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу
на здобуття освітньо-кваліфікаційного рівня «фаховий молодший бакалавр»
студенту Бораку Івану Володимировичу
(прізвище, ім’я та по-батькові студента)

1. Тема роботи Програмний засіб асиметричного шифрування файлів на основі еліптичних кривих

затверджена наказом по коледжу від “ 27 ” листопада 2023 р., №

2. Термін здачі студентом завершеної роботи “___” _____2024 р.

3. Вихідні дані до роботи Розробка програмного засобу асиметричного ширфування файлів на основі еліптичних кривих з використанням бібліотек pycryptodome, tinyec

4. Перелік питань, які повинні бути розроблені:

а) основна частина: встановлення та формалізація вимог, проєктування інтерфейсу, програмна реалізація застосунку, тестування

б) техніко-економічне обґрунтування: аналіз ринку збуту, обґрунтування витрат на проєктування, обґрунтування необхідності розробки

5. Перелік графічного матеріалу алгоритм роботи програми

6. Консультанти роботи: Меленчук Любов Іванівна

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного обґрунтування	_____ (вчена ступень, звання П.І.Б. _____ консультант)		

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми кваліфікаційної роботи.	23.11.2023	01.12.2023
2.	Детальне ознайомлення з предметною областю. Аналіз програмних рішень.	02.04.2024	05.04.2024
3.	Опрацювання теоретичних матеріалів, написання розділу роботи.	06.04.2024	07.04.2024
4.	Формалізація вимог. Аналіз технологій реалізації. Написання розділу роботи.	08.04.2024	15.04.2024
5.	Проєктування та реалізація графічного інтерфейсу. Реалізація основних функцій програми.	18.04.2024	05.05.2024
6.	Тестування програмного засобу.	05.05.2024	16.05.2024
7.	Створення відповідного розділу роботи	16.05.2024	22.05.2024
8.	Обґрунтування вартості проєкту.	26.05.2024	01.05.2024
9.	Оформлення пояснювальної записки.	14.06.2024	14.06.2024
10.	Попередній захист кваліфікаційної роботи.	17.06.2024	17.06.2024
11.	Підготовка до захисту та виправлення помилок.	17.06.2024	24.06.2024
12.	Захист кваліфікаційної роботи.	26.06.2024	26.06.2024

7. Дата видачі завдання “_____” _____ 2023 р. Керівник /_____/

Завдання прийняв до виконання _____/ /

Реферат

Кваліфікаційна робота. Програмний засіб асиметричного шифрування файлів на основі еліптичних кривих. 77 сторінок, рисунків – 23, додатків – 1.

Об’єкт дослідження – процеси шифрування і розшифрування файлів алгоритмом ECC.

Метою роботи є створення програмного засобу для асиметричного шифрування файлів на основі еліптичних кривих мовою Python з використанням бібліотек pycryptodome, tinyec.

Програмний засіб для асиметричного шифрування файлів на основі еліптичних кривих повинен забезпечити генерацію пар ключів (публічного та приватного) з використанням еліптичних кривих, захист і зберігання ключів, шифрування файлів з використанням публічного ключа, розшифрування файлів за допомогою приватного ключа, а також інтеграцію з популярними файловими системами та забезпечення зручного інтерфейсу для користувачів на базі операційних систем Windows, macOS, Linux.

Результатом розробки став програмний засіб асиметричного шифрування файлів на основі еліптичних кривих.

ШИФРУВАННЯ ФАЙЛІВ, ЕЛІПТИЧНІЙ КРИВІ, ECC, WINDOWS, MACOS, LINUX, PYTHON.

Abstract

Qualification work. A software tool for asymmetric file encryption based on elliptic curves. 77 pages, 23 figures, 1 appendix.

The object of the study is the process of file encryption and decryption using the ESS algorithm.

The object of the study is the process of file encryption and decryption using the ESS algorithm.

The purpose of the work is to create a software tool for asymmetric encryption of files based on elliptic curves in Python using pycryptodome, tinyec libraries.

A software tool for asymmetric file encryption based on elliptic curves should provide key pair generation (public and private) using elliptic curves, key protection and storage, file encryption using a public key, file decryption using a private key, as well as integration with popular file systems and providing a convenient interface for users based on Windows, macOS, Linux, Android and iOS operating systems.

The result of the development was a software tool for asymmetric encryption of files based on elliptic curves.

FILE ENCRYPTION, ELLIPTIC CURVES, ECC, WINDOWS, MACOS, LINUX, PYTHON.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ECC – алгоритм асиметричного шифрування на основі еліптичних кривих.

RSA – асиметричний алгоритм шифрування Рівеста-Шаміра-Адлемана.

DSA – алгоритм цифрового підпису.

ЕЦП – електронний цифровий підпис.

ECIES – Elliptic Curve Integrated Encryption Scheme.

ВСТУП

У сучасному цифровому світі, де обмін інформацією в мережі стає все більш невід'ємною частиною нашого повсякденного життя, забезпечення безпеки даних набуває величезного значення. Одним з основних методів захисту конфіденційності інформації є шифрування. Шифрування дозволяє перетворити звичайний текст у нечитабельний для сторонніх осіб вид, що може бути прочитаний лише за допомогою відповідного ключа.

У цьому контексті, асиметричне шифрування виступає як один з найбільш ефективних методів забезпечення безпеки даних. Відмінністю асиметричного шифрування є те, що для шифрування та розшифрування використовуються різні ключі - публічний і приватний, що робить його хорошим вибором для захисту інформації.

На сьогоднішній день найбільш поширеними алгоритмами шифрування є AES, RSA, ECC та Diffie-Hellman Key Exchange. Ці алгоритми забезпечують високий рівень безпеки та використовуються для захисту конфіденційності даних та забезпечення безпечної комунікації в різних сферах, включаючи інтернет-протоколи та мобільні додатки.

Одним із найбільш перспективних підходів до асиметричного шифрування є використання еліптичних кривих. Еліптичні криві пропонують високий рівень безпеки при відносно невеликих розмірах ключів, що робить їх особливо привабливими для застосування в області криптографії.

Метою кваліфіаційної роботи є розробка програмного засобу асиметричного шифрування файлів на основі еліптичних кривих. Дослідження у цьому напрямку є актуальним, оскільки забезпечення безпеки даних є однією з основних проблем у сучасному інформаційному суспільстві.

Поставленим завданням кваліфікаційної роботи є створення програмного засобу асиметричного шифрування та розшифрування файлів на основі еліптичних кривих.

Об’єктом дослідження є процеси шифрування і розшифрування файлів алгоритмом ECC.

Предметом дослідження є існуючі засоби розробки програмних засобів шифрування та розшифрування файлів алгоритмом ECC.

Реалізація поставленої мети передбачає вирішення таких завдань:

– Аналіз предметної області.

– Дослідити інфраструктуру відкритих та закритих ключів для асиметричних криптосистем.

– Проаналізувати асиметричні алгоритми.

– Проаналізувати алгоритм асиметричного шифрування на основі еліптичних кривих.

– Виконати проєктуваня та реалізацію програмного забезпечення.

– Виконати техніко-економічного обґрунтування роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАННЯ

1.1 Опис предметної області

Одним з ключових питань, що виникають в процесі розробки цього проєкту, є забезпечення захисту та конфіденційності інформації, що особливо важливо в сучасному світі. Впровадження цієї системи допомагає підвищити рівень захисту персональних даних, важливих документів, розробки та іншої конфіденційної інформації. Однією з переваг такої системи є простота використання і високонадійний рівень захисту даних.

Алгоритм асиметричного шифрування на основі еліптичних кривих реалізований за допомогою простого математичного підходу, що значно полегшує його реалізацію в структурах з відкритим ключем. Адаптивність і безпека роблять ECC одним з найбільш широко використовуваних алгоритмів підпису в різних програмних додатках, включаючи сертифікати, зашифровані транзакції, електронний підпис повідомлень, а також шифрування і дешифрування файлів і повідомлень.

Система має два основних недоліки. По-перше, існує ризик втрати ключа або самого зашифрованого файлу, що може унеможливити його подальше дешифрування. Іншим недоліком є те, що система працює відносно повільно.

Метою цього дослідження є створення загальнодоступного додатку для забезпечення повного захисту інформації. Це дозволяє перетворювати інформацію з формату, прийнятного для звичайного читання, в формат, який неможливо розшифрувати без використання програмного забезпечення, і навпаки відновлювати інформацію одержувача, що унеможливлює читання без закритого ключа людиною, яка перериває або чує канал зв'язку. Сфера застосування шифрування розширюється і включає не тільки безпечну передачу інформації, а й різні способи перевірки цілісності повідомлень і

ідентифікації відправника або одержувача, цифровий підпис, інтерактивне підтвердження і технології безпечного обміну даними.

З розповсюдженням комп'ютерів та електронних пристроїв стало можливим розробити складну криптографію. Однією з найбільших переваг комп'ютера є те, що він може шифрувати дані, які можуть бути представлені у двійковому форматі. Більшість сучасних комп'ютерних паролів засновані на використанні двійкових бітів. Шифрування на комп'ютерах стало набагато більш ефективним і безпечним.

Шифрування не тільки захищає ваші дані від несанкціонованого доступу. Це можна використовувати, наприклад, для перевірки цілісності та достовірності інформації за допомогою цифрового підпису. Шифрування також є ключовим елементом управління правами доступу та захисту від несанкціонованого копіювання. Крім того, видалену інформацію іноді можна відновити за допомогою спеціального пристрою, щоб її можна було використовувати для безпечного видалення даних. Але якщо дані зашифровані перед видаленням і ключ втрачено, можна відновити лише зашифрований текст, а вихідні дані неможливо відновити.

Багато систем зв'язку використовують шифрування для захисту наших особистих даних від небажаного вторгнення. Ця технологія використовується в різних інших сферах, включаючи системи електронного голосування, інформаційні системи та месенджери.

1.1.1 Використання асиметричних алгоритмів

Шифрування з відкритим ключем, також відоме як асиметричне шифрування, є потужним інструментом для забезпечення безпеки та конфіденційності мережевих комунікацій. Цей тип шифрування використовує пару відкритих та приватних ключів. Відкритий ключ доступний для всіх користувачів і використовується для шифрування повідомлень та перевірки цифрових підписів. Приватний ключ, який є властивістю одержувача,

використовується для дешифрування повідомлення або створення цифрового підпису. Ця схема забезпечує конфіденційність та надійність даних, оскільки неможливо розшифрувати повідомлення без приватного ключа, навіть якщо у вас є відкритий ключ.

Однією з основних проблем, що виникають при використанні асиметричного шифрування, є підтримка безпечного обміну відкритими ключами. Для вирішення цієї проблеми використовуються сертифікати відкритого ключа. Сертифікат – це цифровий документ, що містить відкритий ключ користувача та підпис центру сертифікації, що підтверджує, що цей ключ відповідає певному об'єкту. Органи сертифікації видають сертифікати і забезпечують їх надійність, що забезпечує довіру між користувачами при обміні відкритими ключами.

У 80-х роках основним алгоритмом симетричного шифрування для внутрішнього використання в США був DES (стандарт шифрування даних). Але вже в 90-і роки стали проявлятися основні недоліки цього стандарту. Одним з основних недоліків була довжина ключа, яка становила всього 56 біт. Виявилося, що в умовах постійного підвищення продуктивності комп'ютера цей розмір ключа недостатньо великий. Це зробило DES вразливим до атак, заснованих на використанні всіх можливих варіантів шифрування.

Навпаки, алгоритм відкритого ключа не вимагає секретного каналу для передачі ключа, оскільки відкритий ключ не вважається секретним.

У сучасному світі існує безліч алгоритмів асиметричного шифрування. Найбільш відомими з них є RSA (Рівест-Шамір-Адлеман), DSA (алгоритм цифрового підпису) і ECC (криптографія на еліптичній кривій). Кожен з цих алгоритмів має свої особливості і додатки, але всі вони мають високий рівень безпеки для обміну даними в мережі.

Крім того, шифрування з відкритим ключем дозволяє вирішувати різні завдання, такі як:

– Захист конфіденційності та цілісності даних.

– Надати цифровий підпис для автентифікації та безперервності.

– Розподілений обмін секретними ключами для забезпечення конфіденційності даних.

– Шифрування великої за обсягом інформації без попередньої зміни закритого ключа.

При шифруванні з відкритим ключем важливо переконатися, що певний відкритий ключ належить конкретному користувачеві. Зазвичай відкриті ключі зберігаються у спільних каталогах відкритих ключів, тому існує ризик їх перехоплення. Для забезпечення такої надійності використовуються механізми, засновані на сертифікатах з відкритим ключем.

Сертифікат відкритого ключа забезпечує надійний зв'язок між відкритим ключем і об'єктом, якому належить відповідний закритий ключ. Цифровий документ містить відкритий ключ суб'єкта та підпис, виданий центром сертифікації. Крім того, сертифікат містить інформацію про власника відкритого ключа, яка додатково ідентифікує власника відкритого ключа.

Сертифікат-це цифровий документ, що підтверджує особу особи, об'єкта чи об'єкта в Інтернеті. До них відносяться відкритий інформаційний ключ і цифровий підпис відповідного центру сертифікації. Основна функція сертифіката-забезпечення безпеки електронних транзакцій, включаючи забезпечення конфіденційності, цілісності та надійності даних.

На сьогоднішній день найбільш поширеною сертифікацією є стандарт міжнародного союзу електрозв'язку ITU-T X. сертифікати відповідають рекомендаціям 509v3 і цільової групи з інженерного забезпечення Інтернету RFC2459. Центр сертифікації виступає в якості гаранта зв'язку між відкритим ключем суб'єкта та інформацією, що міститься в сертифікаті. Різні органи з сертифікації встановлюють і гарантують ці відносини на різних принципах. Це

вимагає, щоб ви ознайомилися з Політикою та правилами, перш ніж довіряти сертифікату.

Органи з сертифікації є важливим компонентом систем безпеки в інформаційних системах з розподіленою структурою.

1.1.2 Електронний цифровий підпис

Електронні цифрові підписи (ЕЦП) – це вимоги до шифрування, які захищають електронні документи від підробки та гарантують справжність їх походження. ЕЦП досягається шляхом шифрування інформації за допомогою приватного ключа ЕЦП, який може ідентифікувати власника сертифіката ключа підпису та визначити, що інформація в електронному документі не була спотворена.

ЕЦП відноситься до протоколу автентифікації, оскільки гарантує, що повідомлення отримано від довіреного відправника. Це також протокол перевірки цілісності, оскільки він гарантує, що повідомлення надходять незмінними.

Електронний цифровий підпис – це бітовий рядок, отриманий в результаті процесу створення підпису (згідно ISO / IEC 14888-1: 2008). Згідно з ДСТУ, терміни "електронний цифровий підпис", "електронний підпис" та "цифровий підпис" можуть використовуватися як взаємозамінні. Ця схема використовується для забезпечення цілісності та надійності електронних документів.

У процесі створення електронного підпису відправник:

– Застосовує хеш-функцію до оригінального повідомлення та отримує хеш-код повідомлення.

– Розраховує електронний підпис на основі отриманого хеш-коду з використанням його закритого ключа.

– Надсилає одержувачу повідомлення разом з електронним підписом.

– Покупець, у свою чергу:

– Застосовує хеш-функцію до отриманого повідомлення та отримує хеш-код повідомлення.

– Використовує відкритий ключ відправника для дешифрування хеш-коду в електронному підписі.

– Перевіряє відповідність отриманого змішаного зображення і, якщо воно збігається, впевнюється, що відправником є той, за кого він себе видає, і що повідомлення не було пошкоджено.

1.2 Аналіз існуючих рішень

Асиметричне шифрування на основі еліптичних кривих – це один з найефективніших способів забезпечення інформаційної безпеки в сучасному світі. При підході до аналізу програмних рішень для цього типу шифрування варто враховувати кілька аспектів: безпеку, ефективність, простоту використання, розгортання і так далі. У цьому аналізі будуть оглянуті деякі популярні програмні інструменти для асиметричного шифрування на основі еліптичних кривих і будуть визначатися їх плюси і мінуси.

Однак, після ретельного пошуку не вдалось знайти додаток, що може шифрувати файли на основі еліптичних кривих. Натомість, існує багато бібліотек, що допоможуть в реалізації такого. Бібліотеки надають інструменти та компоненти, необхідні для розробки, але самі по собі не є готовими продуктами для кінцевого користувача. Вони вимагають додаткової розробки та інтеграції в існуючі системи або створення нових додатків з їх використанням. Було оглянуто 3 популярні бібліотеки для криптографії. Результати огляду наведено в наступних пунктах.

1.2.1 OpenSSL

OpenSSL – один з найбільш широко використовуваних криптографічних пакетів з відкритим кодом, який підтримує широкий спектр алгоритмів, включаючи асиметричне шифрування на основі ECC.

Основними перевагами OpenSSL є:

– Відкритий код, який дозволяє перевірити безпеку власного додатку та вносити зміни у нього.

– Широкий функціонал, який має підтримку великого спектру алгоритмів шифрування, включаючи ECC.

– Широка підтримка, що дозволяє використовувати бібліотек на багатьох платформах.

Однак у OpenSSL є деякі недоліки:

– Складність використання, оскільки інтерфейс командного рядка може бути складним для новачків.

– Через велику кількість програмного коду можуть виникати серйозні помилки.

1.2.2 Bouncy Castle

Bouncy Castle – це бібліотека алгоритмів шифрування, написана на Java, а також підтримує асиметричне шифрування на основі ECC.

Основні переваги включають:

– Компактність, оскільки Bouncy Castle містить менший обсяг коду в порівнянні з іншими бібліотеками.

– Підтримка мови Java. Дана бібліотека написані на цій мові, що робить її привабливою для розробників, що працюють з нею.

Основні недоліки:

– Обмежена підтримка мов. Хоча й Bouncy Castle підтримує мову Java, але інтеграція з іншими мовами може бути складною. Це робить Bouncy Castle

менш привабливою для проєктів, що використовують інші мови програмування.

1.2.3 Crypto++

Crypto++ – це ще одна бібліотека відкритих алгоритмів шифрування, яка підтримує асиметричне шифрування на основі еліптичних кривих. Вона написана на C++, тому вона приваблює розробників, які використовують цю мову програмування.

Основні переваги:

– Широкий спектр алгоритмів, що підтримуються, тому розробники можуть вибрати алгоритм, який найкраще відповідає їхнім потребам.

– Відкритий код, який дозволяє контролювати безпеку додатку та вносити власні зміни.

Основні недоліки:

– Crypto++ може бути складним у використанні для початківців та розробників, які не мають досвіду роботи з C++.

– Crypto++ має обмежену підтримку мов програмування, що може обмежити його привабливість для деяких проєктів.

Шифрування на основі ECC залишається активною сферою досліджень шифрування. У майбутньому очікується розробка нових програмних засобів з поліпшеними характеристиками безпеки і продуктивності. Також важливо продовжувати перевіряти існуючі рішення, щоб виявити та виправити потенційні вразливості.

1.3 Аналіз вимог та постановка завдання

Аналіз вимог до проєкту є важливим, оскільки він дозволяє чітко зрозуміти потреби клієнта, зменшити ризики, ефективно планувати ресурси, бюджет і час, а також уникнути значних змін під час виконання проєкту. Це забезпечує відповідність кінцевого продукту очікуванням користувачів,

сприяє своєчасному виконанню завдань та допомагає уникнути зайвих витрат і затримок.

Під час аналізу вимог до проєкту слід враховувати функціональні та нефункціональні вимоги, аби готовий проєкт забезпечував повноту роботи та був ефективним.

Функціональні вимоги до проєкту:

– Генерування ключів на основі еліптичної кривої та їх збереження у файли.

– Надати можливість вибору ключів, що вже існують.

– Забезпечити можливість обирати файл для шифрування.

– Підтримка файлів із різним розширенням.

– Збереження розшифрованого файлу з оригінальним розширенням.

– Програма повинна дати змогу ввести текст та одразу його зашифрувати.

Нефункціональні вимоги:

– Забезпечити ефективну роботу програми.

– Реалізація простого та зрозумілого користувацького інтерфейсу.

– Забезпечити коректність шифрування та розшифрування файлів.

Аналіз вимог є критично важливим для успішної реалізації проекту. Він забезпечує розуміння того, що необхідно створити, які ресурси потрібні та як досягти поставлених цілей. Правильно проведений аналіз вимог допомагає уникнути багатьох проблем на подальших етапах розробки та впровадження проекту.

2 АЛГОРИТМІЧНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Еліптична криптографія

Еліптична криптографія – це галузь шифрування, яка використовує властивості еліптичної кривої для генерації криптографічних протоколів, алгоритмів шифрування та підписів. Вона заснована на математичній теорії еліптичних кривих, яка дозволяє вивчати властивості точок на кривих і використовувати їх для вирішення криптографічних задач, відрізняється від традиційних методів, таких як RSA, тим, що заснований на додаванні і множенні точок на еліптичних кривих, а не складна арифметика над простими числами. Це дозволяє скоротити обсяг операцій з ключами і шифруванням, забезпечуючи при цьому високий рівень безпеки[2].

Еліптична криптографія була вперше представлена Віктором Міллером та Нілом Кобліцем у 1985 році. Безпека цього методу базується на проблемі дискретної логарифмічної еліптичної кривої (ECDLP). ECC можна визначити за допомогою ряду параметрів($q,\ a,\ b,\ G,\ n,\ h$). Цей метод має 2 важливі переваги, які особливо ефективні при обмежених обчислювальних ресурсах:

– Для забезпечення того ж рівня безпеки ECC вимагає набагато меншого розміру ключа в порівнянні з іншими системами шифрування з відкритим ключем, такими як RSA.

– Операції скалярного множення в ECC набагато швидші, ніж модульні показники, і їх можна легко реалізувати на обладнанні.

Еліптична криптографія використовує асиметричну криптографію, засновану на математичних властивостях еліптичних кривих. У цій схемі у кожного користувача є 2 ключа: відкритий і закритий.

Відкритий ключ, який використовується для шифрування даних або перевірки цифрового підпису. Він може бути доступним для всіх, щоб кожен міг отримати до нього доступ. Зазвичай відкритий ключ називається

публічним ключем, оскільки він може бути опублікований для зв'язку з іншими користувачами.

Закритий ключ, який використовується для дешифрування даних або створення цифрового підпису. Він повинен залишатися приватним і бути доступним лише власнику ключа. Зазвичай закритий ключ називається приватним ключем і використовується для виконання конфіденційних операцій.

2.1.1 Схема Діффі-Хеллмана

Протокол ECDH є варіантом протоколу Діффі-Хеллмана, який використовується для еліптичної криптографії. Його основна мета – безпечно обмінюватися спільними ключами для симетричного шифрування через незахищений канал зв'язку.

Розглянемо класичний приклад використання декомпозиції цього Протоколу між 2 користувачами на ім'я Аліса та Боб, які спілкуються через незахищений канал зв'язку:

Аліса і Боб використовують одну і ту ж базову точку G для створення пар відкритих і приватних ключів для створення підгруп еліптичних кривих.

r_b – закритий ключ Боба;

$H_b = r_b * G$ – відкритий ключ Боба;

r_a – закритий ключ Аліси ;

$H_\text{a} = r_\text{a} * G$ – відкритий ключ Аліси.

Вони обмінюються відкритими ключами через незахищені канали. Навіть якщо зловмисник перехоплює ці відкриті ключі, проблему дискретного логарифму потрібно вирішити, щоб отримати відповідний приватний ключ.

Аліса та Боб обчислюють спільний відкритий ключ, який використовується для створення спільного секретного ключа.

$$U = r_a * H_b = r_a(r_b * G) = r_b(r_a * G) = r_b * H_a.$$

В разі перехоплення зловмисником відкритих ключів Аліси та Боба, проблему дискретного логарифму важко вирішити.

Отримавши спільний відкритий ключ, Аліса та Боб можуть використовувати симетричне шифрування, таке як алгоритм AES, для безпечного спілкування.

2.2 Еліптична крива

Еліптичні криві є одним з основних об'єктів вивчення в сучасній теорії чисел і криптографії. Еліптична криптографія є самостійним розділом криптографії, що присвячений вивченню криптосистем на базі еліптичних кривих[1].

Еліптичну криву E можна визначити як алгебраїчну криву, що задовольняє наступному рівнянню:

$$y^2 = x^3 + ax + b.$$

Для цілей шифрування використовуються тільки неособисті еліптичні криві. Еліптична крива називається безособовою, якщо дискримінант λ не дорівнює нулю, тобто:

$$\Delta = 4a^3 + 27b^2 \neq 0.$$

На рисунку 2.1 показано 2 види еліптичних кривих:

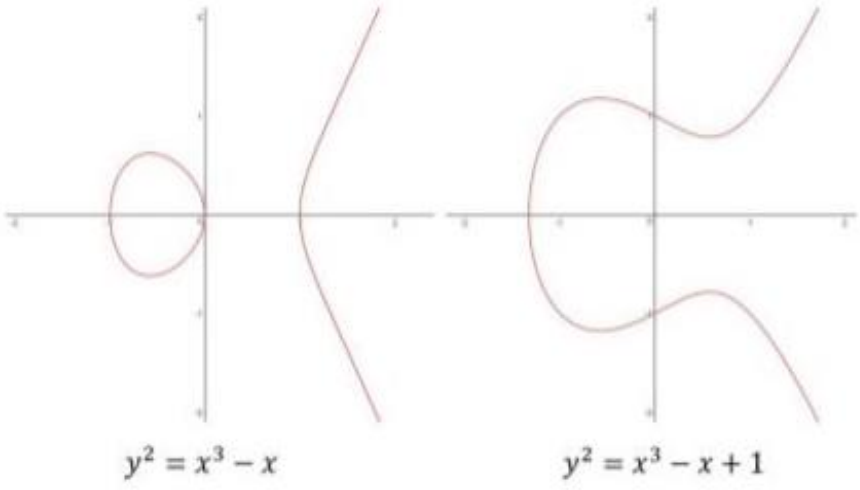

Рисунок 2.1 – Види еліптичних кривих

Далі проаналізуємо математичні операції, що можна виконати над точками на еліптичній кривій.

2.2.1 Груповий закон

Груповий закон – це правила, які визначають, як елементи набору груп об'єднуються для створення нових елементів. Він може бути представлений як операція об'єднання 2 елементів групи або як більш складна процедура визначення способу об'єднання більшої кількості елементів.

Наприклад, для групи цілих чисел з операцією додавання (+) груповий закон визначає, що сума 2 цілих чисел також є цілим числом. У групі обертання одного кола за допомогою процесу формування кута Груповий закон визначає, як кути повинні бути з'єднані за допомогою композиції для отримання нового кута.

Групою називається впорядкована пара <G, *>, де множина G не є порожньою, і виконуються такі умови:

– (*) – бінарна алгебраїчна операція задана на множині G.

– Операція * є асоціативною для G, тобто для будь-якого елемента множини G – a,b,c справедливе правило $(a * b) * c = a * (b * c)$.

– В G є нейтральний відносно операцій * елемент e, при якому

$a * e = e * a = a$ для всіх $a \in G$.

– Для будь-якого елемента $a \in G$ існує симетричний до нього елемент $a' \in G$, при якому $a * a' = a' * a = e$.

2.2.2 Додавання точок

Процес додавання точок на еліптичній кривій виконується шляхом додавання двох заданих точок P і Q. Цей процес призводить до третьої точки R на тій же еліптичній кривій. Спочатку проводиться лінія, що з'єднує ці дві точки. Ця лінія перетинаю еліптичну криву ще у третій точці, яку прийнято позначати -R(x, -y). Подальше відображення точки через вісь дає точку R(x, y)

Процес додавання точок зазвичай позначається як " R = P + Q ". Результат додавання точок можна побачити на рисунку 2.2.

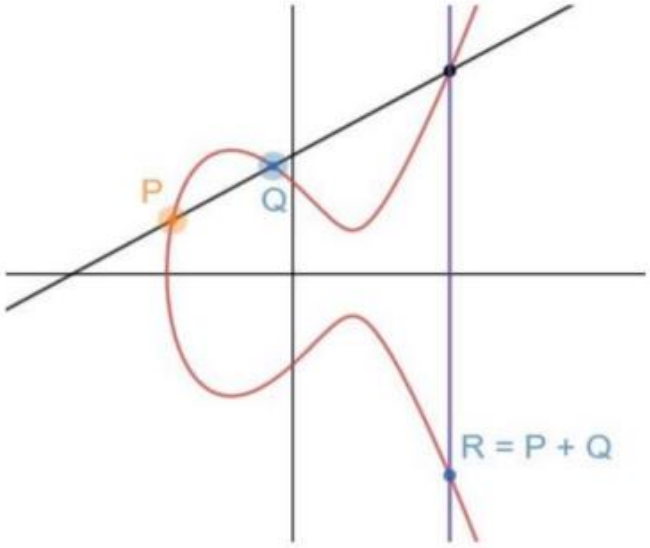

Рисунок 2.2 – Додавання точок

2.2.3 Подвоєння точок

Процес подвоєння точок на еліптичній кривій, показаний на рисунку 2.3, включає в себе додавання точки P до самої себе на кривій. Це схоже на процес додавання точок, за винятком того, що в цьому випадку у нас немає двох точок для визначення лінії. Натомість ми беремо дотичну до еліптичної кривої в точці P, і ця дотична перетинає еліптичну криву-R(x, y) в іншій точці тієї ж

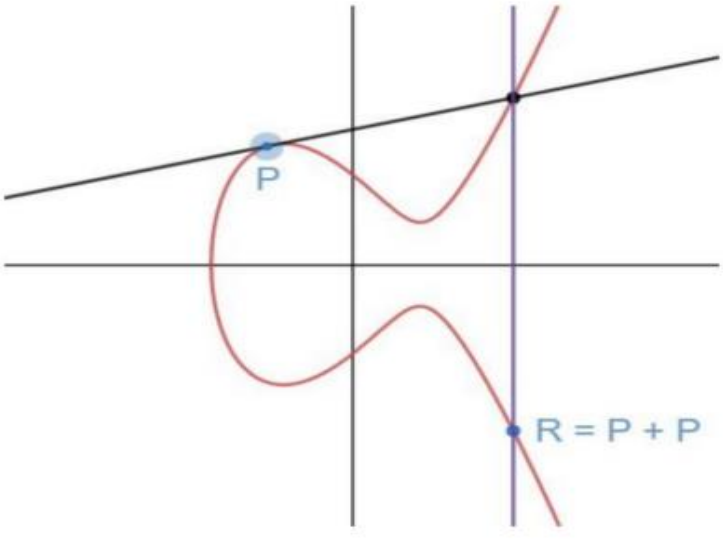

Рисунок 2.3 – Подвоєння точок

еліптичної кривої. Подальше відображення R (x, y). Процес подвоєння точки зазвичай позначається як " R = P + P " (рисунок 2.3).

2.2.4 Множення точки

Щоб помножити точку на еліптичній кривій, потрібно помножити скалярне значення на точку, в результаті чого вийде інша точка на тій же еліптичній кривій. Це досягається шляхом послідовного застосування операції множення на 2 і додавання балів. Наприклад, якщо P - точка, а n - скалярне значення, яке потрібно помножити на точку, то nP приймає n помножених як суму P і самого себе. Однак основний підхід полягає в тому, що точка P додається до себе n разів, і це можна записати як $nP = P + P + ... + P$ (n разів).

2.3 Аналіз криптостійкості ECC

Перевірка стабільності алгоритму ECC полягає в ускладненні проблеми дискретного логарифмування на еліптичних кривих, яка є основним принципом шифрування ECC. На відміну від RSA, де стабільність залежить від складності факторизації великих чисел, ECC використовує закон асоціативного додавання точок на еліптичній кривій, що значно ускладнює криптографічну роботу.

Методи атаки на ECC включають використання алгоритму для пошуку перетину еліптичних кривих зсередини себе, пошуку кратних точок та використання алгоритму декомутації для даної кривої. Однак алгоритм відносно стабільний порівняно з RSA, оскільки навіть найвідоміші атаки на ECC вимагають експоненціального часу для їх виконання.

Порівняно з RSA, атаки на ECC з використанням методів факторизації менш ефективні, оскільки не існує ефективного алгоритму для факторизації великих чисел. Більшість атак на ECC базуються на еліптичних кривих та математичних властивостях їх точок.

Алгоритми ECC, такі як RSA, є стандартом для шифрування в багатьох протоколах і системах. Його використання рекомендується інформаційними

стандартами та правилами і широко використовується в різних галузях, включаючи Інтернет-комунікації, електронну комерцію та мобільну безпеку.

2.4 Проєктування схеми шифрування з еліптичною кривою

Для реалізацій програмного застосунку буде використовуватися інтегрована схема шифрування з еліптичною кривою. Інтегрована схема шифрування з еліптичною кривою (ECIES) – це криптографічна схема, яка поєднує в собі властивості асиметричного та симетричного шифрування з використанням еліптичних кривих.

Основні етапи ECIES:

1) Генерація ключів.

– Призначення ключа починається зі створення параметрів для еліптичної кривої та базової точки на цій кривій. Потім створюється симетричний ключ шифрування, який використовується для обміну конфіденційними даними.

2) Шифрування.

– Випадково згенерований одноразовий ключ для симетричного шифрування.

– Відкритий ключ генерується за допомогою отриманих параметрів еліптичної кривої та отриманого випадкового ключа.

– Текст повідомлення шифрується за допомогою симетричного шифрування з випадковими ключами. Випадкові ключі також шифруються за допомогою відкритих ключів.

3) Розшифрування:

– Отриманий зашифрований текст розшифровується за допомогою відкритого ключа.

– Розшифрований текст буде містити зашифрований випадковий ключ і зашифрований текст повідомлення.

– Зашифрований випадковий ключ розшифровується за допомогою власного приватного ключа.

– Розшифрований випадковий ключ використовується для розшифрування зашифрованого тексту.

Демонстрацію алгоритму шифрування файлів можна побачити на рисунку 2.4.

Рисунок 2.4 – Алгоритм шифрування файлів

На рисунку 2.5 зображено алгоритм розшифрування зашифрованих файлів

Рисунок 2.5 – Алгоритм розшифрування файлів

За допомогою ECIES можна використовувати відкритий ключ одержувача для шифрування ваших даних, забезпечення конфіденційності та використання випадкових ключів для симетричного шифрування для

забезпечення ефективності та безпеки. Ця схема дозволяє використовувати переваги еліптичного шифрування для забезпечення безпеки обміну даними.

2.4.1 Область визначення алгоритму

Еліптична криптографія заснована на використанні циклічних підгруп еліптичних кривих над кінцевими полями. Параметри алгоритму включають наступне:

– Коефіцієнт еліптичної кривої (a, b).

– Просте число (p), що визначає порядок кінцевого поля.

– Точка, що утворює підгрупу (G).

– Порядок і кофактор цієї підгрупи (n, f).

Важливо зазначити, що існують певні класи еліптичних кривих, які вразливі до атак і можуть бути атаковані за допомогою алгоритмів дискретного логарифмування за поліноміальний час.

Наприклад, у вас є клас еліптичних кривих, де порядок підгруп і добуток кофакторів дорівнюють порядку полів, що робить їх вразливими до інтелектуальних атак. У таких випадках важливо переконатися, що крива спеціально не призначена для захисту від атак. Для цього використовуйте початковий параметр (початкове значення), який використовується для визначення коефіцієнтів рівняння еліптичної кривої. Зазвичай це випадкове значення, отримане за допомогою функції випадкового вибору. Потім хеш-функція обчислюється з цього числа, і параметри кривої залежать від цього хешу.

$$S = rand()$$

$$H = hash(S)$$

$$a = m(H)$$

$$b = n(H)$$

Використовуючи незворотність хеш-функції, гарантується, що параметри еліптичної кривої не будуть обрані для приналежності до класу слабких еліптичних кривих.

Слід зазначити, що еліптичні криві, стандартизовані Американським інститутом стандартів і технологій, можуть бути побудовані з неперевіреними параметрами, залишаючи питання, пов'язані з безпекою. Навпаки, RSA може здатися більш надійним, оскільки не вимагає спеціальних полів визначення параметрів. Однак у цьому випадку важливо, щоб ви могли створити власні налаштування безпеки.

2.4.2 Алгоритм генерування відкритого та закритого ключів ECC

У криптографічних системах, заснованих на еліптичних кривих над кінцевими полями, є спеціально обрана точка, яка називається твірною точкою G або твірною точкою. Цю точку можна побудувати, помноживши G на ціле число в грудні [0…r] до будь-якої іншої точки цієї підгрупи на еліптичній кривій. Тут число r називається" порядком " циклічної підгрупи і представляє загальну кількість точок у підгрупі.

Для еліптичної кривої, кофактор якої дорівнює 1, існує лише одна підгрупа, а ступінь кривої n (включаючи всі різні точки на кривій, включаючи нескінченність) дорівнює числу r. Якщо точка G і порядок n ретельно обрані, всі можливі точки на еліптичній кривій, включаючи нескінченність, будуть отримані шляхом множення генеруючої точки G на цілі числа в діапазоні від 1 до n. Це ціле число n відоме як "порядок кривих".

Порядок r підгрупи визначається точкою G (конструктор еліптичної кривої (EC)) і може відрізнятися від порядку кривої. Послідовність обчислюється як відношення порядку кривої до кофактора і визначає загальну кількість можливих секретних ключів для даної кривої. Криптографи ретельно вибирають параметри еліптичної кривої, такі як рівняння кривої, точки генерації та кофактори. Він забезпечує великий простір ключів для задоволення вимог до криптографічної потужності.

В криптографія еліптичної кривої (ECC), точки на еліптичній кривій разом із генеруючою точкою G утворюють циклічну групу, яка називається

циклічною підгрупою. Це пов'язано з тим, що існує певне число r (де r> 1), де r * G = 0 * G = нескінченність, і всі точки цієї підгрупи генеруються шляхом множення точки генерації G від 1 до r на ціле число.

Підгрупи еліптичних кривих зазвичай мають багато потенційних точок формування. Однак криптографи ретельно вибирають один із способів, за допомогою якого можуть бути створені цілі групи або підгрупи для оптимізації обчислювальної продуктивності.1 це вибране значення називається "G".

На деяких еліптичних кривих різні точки формування можуть давати підгрупи різного ступеня. Іншими словами, якщо група має порядок n, то для кожного простого числа d дільник n, є точка Q такий як d * Q= нескінченність. Це означає, що певні точки, що використовуються як будівельники для певних кривих, утворюють менші підгрупи, ніж інші. Якщо розмір групи невеликий, атаки "невеликих підгруп" можуть поставити під загрозу безпеку. Це одна з причин, чому криптографи зазвичай вибирають порядок підгрупи r як просте число 1.

Для еліптичної кривої з кофактором h > 1 різні базові точки можуть утворювати різні підмножини точок EC на кривій. Вибравши Генератор точок, ви можете працювати з певною підмножиною точок на кривій, використовуючи більшість алгоритмів шифрування точок EC, але в деяких випадках слід бути особливо обережними і рекомендується використовувати лише перевірені програми, алгоритми та програмні засоби ECC.

У криптографії еліптичної кривої (ECC) множення фіксованої точки кривої (відомої як точка генерації) на задане ціле число k дає іншу точку P на тій самій еліптичній кривій. Це число K можна розглядати як приватний ключ, але точка P вважається відповідним відкритим ключем.

Отже, у криптографії з еліптичною кривою це виглядає так:

– Еліптична крива (EC) на кінцевому полі F_p.

– G – це точка генерації, яка є постійною і є базовою точкою на еліптичній кривій.

– k – приватний ключ, який є цілим числом.

– p – це відкритий ключ, точка на еліптичній кривій, отримана шляхом множення G на K .

Ефективні обчислення для P = G * k зазвичай виконуються дуже швидко, використовуючи добре відомий алгоритм множення еліптичних кривих, який виконується з часом пропорційно логарифму K.

Такі обчислення для 256-бітових кривих виконуються дуже швидко, оскільки для них потрібно лише кілька сотень простих операцій ЄС. Однак, операція обчислення k = P / g вважається дуже повільною і практично нездійсненною для великих значень k.

Ця асиметрія між швидким множенням і повільним діленням є основою безпеки шифрування еліптичних кривих (ECC), відомого як проблема дискретного логарифму на еліптичних кривих (ECDLP).

В інформатиці проблема дискретної логарифмічної еліптичної кривої (ECLP) визначається як:

Завдання полягає в тому, щоб знайти ціле число k (якщо воно існує), а добуток точки G на кривій, відомий як точка генерації на k, також дорівнює точці P, розташованій на цій кривій. Іншими словами, ми шукаємо k таким чином, щоб p = G * K.

Для ретельно відібраних кінцевих полів та еліптичних кривих, що використовуються криптографами, завдання ECLP вважається громіздким. Це означає, що якщо система налаштована правильно, немає ефективного способу швидко знайти цей тип k, особливо для великих значень k .

Кратність точок на еліптичній кривій в групі чисел дорівнює кількості чисел з точністю до ступеня групи чисел, а задача ECDLP аналогічна задачі дискретного логарифма (DLP).

У криптографії еліптичних кривих (ECC) багато алгоритмів базуються на обчислювальній складності задач ECDLP з ретельно підібраними областями F_p та еліптичними кривими без правильного алгоритму рішення.

Найшвидший відомий алгоритм вирішення задач ECDLP для ключів розміру k вимагає n кроків, тому для досягнення криптографічної стабільності на рівні k біт, наприклад, на 256-бітовій кривій, 512-бітове поле F_p зазвичай використовується для забезпечення криптостійкості близько 128 біт.

Однак порядок (n) кривої, як правило, менший за розмір поля (p), і крива може мати кофактор h> 1, тому фактична надійність пароля може бути нижчою. Це призводить до того, що порядок підгрупи r = n / h менше, ніж наступний n. Враховуючи, що кількість кроків в алгоритмі становить не тільки k, але і близько 0,886 * k. Точну оцінку криптографічної сили різних еліптичних кривих можна знайти в наукових дослідженнях.

Наприклад, еліптична крива secp256k1 (p = 256) забезпечує захист приблизно при 128 бітах (точніше, 127,8 біт), тоді як крива 448 (p = 448) забезпечує захист приблизно при 224 бітах (точніше, 222,8 біт).

2.5 Проєктування графічного інтерфейсу

Розробка графічного інтерфейсу користувача (GUI) є важливим процесом розробки програмного забезпечення, який забезпечує зручність використання і естетичну привабливість програми. Графічний інтерфейс дозволяє користувачеві взаємодіяти з додатком за допомогою візуальних елементів, таких як кнопки, меню, піктограми, вікна та інші графічні компоненти.

Завдання полягає в проєктуванні простого та інтуїтивно зрозумілого графічного інтерфейсу, яким зможе користуватися навіть не дуже впевнений

користувач. Також інтерфейс має надавати доступ до всіх функцій програмного застосунку.

Інтерфейс повинен містити наступні елементи:

– Текстове поля для введення тексту.

– Кнопку для генерації ключів.

– Кнопку для збереження ключів.

– Кнопку для вибору ключів.

– Кнопку для шифрування файлів.

– Кнопку для збереження та шифрування щойно введеного тексту.

– Кнопку для розшифрування тексту.

Такий функціонал графічного інтерфейсу надасть можливість користуватися всім функціонал програмного застосунку.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Реалізація графічного інтерфейсу

При проєктуванні графічного інтерфейсу були поставлені вимоги створити зручний, інтуїтивно зрозумілий інтерфейс, який буде надавати доступ до всіх функції програмного застосунку.

На основі поставлених вимог був спроєктований наступний графічний інтерфейс, як приведено на рисунку 3.1.

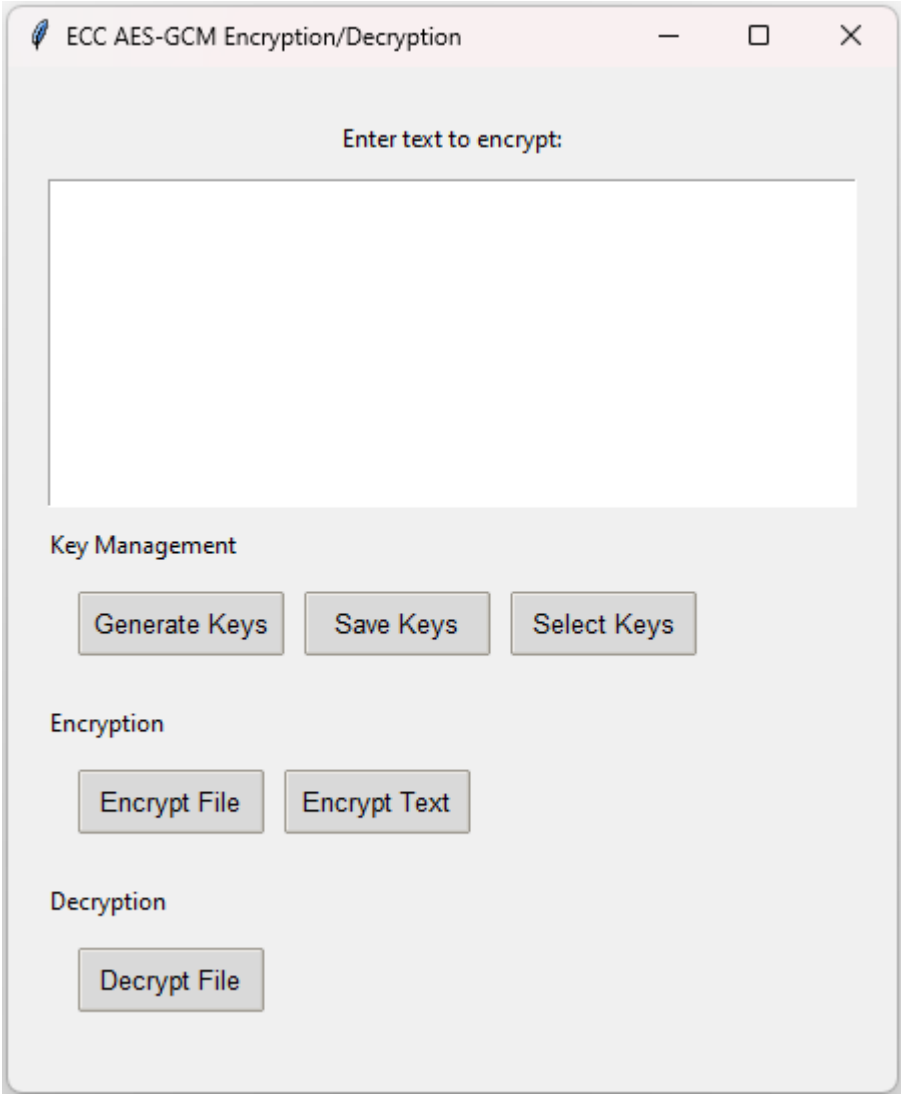

Рисунок 3.1 – Графічний інтерфейс

Він містить набір усіх потрібних елементів, зазначених у вимогах. Для його реалізацій була застосована бібліотека tkInter для Python, а реалізовано все в середовищі Visual Studio Code від Microsoft[7]. Visual Studio Code (VS Code) є досить зручним та швидким засобом розробки як для невеликих

проєктів, так і для корпоративних рішень. Це середовище надає розробникам можливості, які дозволяють ефективно працювати з різноманітними мовами програмування та інструментами. Оскільки шифрування файлів асиметричними алгоритмами потребує високої швидкодії коду, було обрано саме VS Code, який забезпечує можливість компілювати код під певну платформу. Завдяки потужним функціям редагування, відлагодження та інтеграції з системами контролю версій, Visual Studio Code допомагає створювати продуктивні та надійні програмні рішення.

Далі буде надана реалізація основних елементів інтерфейсу.

Реалізація поля для введення тексту наведена в лістингу 3.1.

Лістинг 3.1 – Поле для введення тексту

```
text_input = tk.Text(frame, height=10, width=50)
text_input.grid(row=1, column=0, columnspan=2, pady=5)
```

Реалізація кнопки для генерації ключів наведена в лістингу 3.2.

Лістинг 3.2 – Кнопка для генерації ключів

```
generate_keys_button = ttk.Button(key_frame, text="Generate Keys", command=generate_keys)
generate_keys_button.grid(row=0, column=0, pady=5, padx=5)
```

Реалізація кнопки збереження ключів наведена в лістингу 3.3.

Лістинг 3.3 – Кнопка для збереження ключів

```
save_keys_button = ttk.Button(key_frame, text="Save Keys", command=save_keys)
save_keys_button.grid(row=0, column=1, pady=5, padx=5)
```

Реалізація кнопки вибору ключів наведена в лістингу 3.4.

Лістинг 3.4 – Кнопка для вибору ключів

```
select_keys_button = ttk.Button(key_frame, text="Select Keys", command=select_keys)
select_keys_button.grid(row=0, column=2, pady=5, padx=5)
```

Реалізація кнопки шифрування файлу наведена в лістингу 3.5.

Лістинг 3.5 – Кнопка для шифрування файлу

```
    encrypt_button = ttk.Button(encryption_frame, text="Encrypt
File", command=encrypt_file)
    encrypt_button.grid(row=0, column=0, pady=5, padx=5)
```

Реалізація кнопки шифрування введеного тексту наведена в лістингу 3.6.

Лістинг 3.6 – Кнопка для шифрування тексту з текстового поля

```
    encrypt_text_button = ttk.Button(encryption_frame,
text="Encrypt Text", command=encrypt_text)
    encrypt_text_button.grid(row=0, column=1, pady=5, padx=5)
```

Реалізація кнопки розшифрування файлу наведена в лістингу 3.7.

Лістинг 3.7 – Кнопка для розшифрування файлу

```
    decrypt_button = ttk.Button(decryption_frame, text="Decrypt
File", command=decrypt_file)
    decrypt_button.grid(row=0, column=0, pady=5, padx=5)
```

3.2 Реалізація основних функцій

Алгоритм асиметричного шифрування на основі еліптичних кривих (ECC) базується на складності задачі дискретного логарифмування на еліптичних кривих, що робить його дуже ефективним і безпечним. Однак, шифрування великих обсягів даних за допомогою ECC може бути менш ефективним через обмеження розміру повідомлення. Тому часто використовують гібридні криптосистеми, де ECC використовується для безпечного обміну симетричним ключем, а потім симетричний алгоритм, наприклад AES-GCM, використовується для шифрування самих даних.

У програмі, ключі генеруються на основі параметрів кривої BrainpoolP256r1. Приватний ключ представляє собою випадкове число, а публічний ключ — точка на кривій, отримана множенням приватного ключа на генератор кривої.

Шифрування повідомлень здійснюється за допомогою функції encrypt_ECC, яка генерує випадковий сеансовий ключ, обчислює спільний секретний ключ за допомогою множення цього ключа на публічний ключ одержувача, а потім використовує цей секретний ключ для симетричного

шифрування повідомлення за допомогою AES-GCM. Для кожного сеансу генерується унікальний вектор ініціалізації (nonce), що забезпечує додаткову безпеку[3].

Розшифрування повідомлень здійснюється за допомогою функції decrypt_ECC, яка використовує приватний ключ одержувача для обчислення спільного секретного ключа, який потім використовується для розшифрування повідомлення за допомогою AES-GCM. Цей процес є безпечним і ефективним, оскільки використання ECC забезпечує високий рівень безпеки при меншому розмірі ключів порівняно з RSA.

Програма також включає функції для зберігання та завантаження ключів у файли, а також для зчитування та збереження даних у файли. Це дозволяє зручно працювати з ключами та зашифрованими повідомленнями, забезпечуючи високу гнучкість і зручність у використанні.

Таким чином, використання ECC для обміну симетричними ключами та AES-GCM для шифрування даних забезпечує ефективну та безпечну гібридну криптосистему, яка поєднує переваги обох алгоритмів[4].

Реалізовані функції будуть наводитися нижче разом із інформацією про дані, які вона приймає та дані, які повертає. Такий спосіб подання інформації допомагає краще зрозуміти структуру програмного коду для майбутньої роботи з ним[5].

Функція save_key_to_file(key, filename) зберігає ключ у файл. Приймає ключ у вигляді цілого числа та ім'я файлу, куди зберігати ключ. Записує ключ у файл у вигляді байтів. Реалізація даної функції наведена в лістингу 3.8.

Лістинг 3.8 – Функція збереження ключів у файли

```
def save_key_to_file(key, filename):
      with open(filename, 'wb') as f:
          f.write(int.to_bytes(key, 32, 'big'))
```

Функція load_key_from_file(filename) завантажує ключ з файлу. Приймає ім'я файлу, з якого зчитувати ключ. Читає вміст файлу та повертає ключ у вигляді цілого числа. Реалізація даної функції наведена в лістингу 3.9.

Лістинг 3.9 – Функція для завантаження ключів з файлу

```
def load_key_from_file(filename):
    with open(filename, 'rb') as f:
        key = int.from_bytes(f.read(), 'big')
    return key
```

Наступною функцією є read_plaintext(input_file). Вона зчитує текстовий файл. Приймає шлях до файлу, читає вміст файлу та повертає його у вигляді байтів. Реалізація даної функції наведена в лістингу 3.10.

Лістинг 3.10 – Функція для зчитування вмісту файлу

```
def read_plaintext(input_file):
     with open(input_file, 'rb') as f:
        return f.read()
```

Функція save_to_file(data, filename) зберігає дані у файл. Приймає дані у вигляді байтів та ім'я файлу, куди їх зберегти. Записує дані у вказаний файл. Реалізація даної функції наведена в лістингу 3.11.

Лістинг 3.11 – Функція для збереження даних у файл

```
def save_to_file(data, filename):
    with open(filename, 'wb') as f:
        f.write(data)
```

Функція load_from_file(filename) завантажує дані з файлу. Приймає ім'я файлу, з якого потрібно зчитати дані. Читає вміст файлу та повертає його у вигляді байтів. Реалізація даної функції наведена в лістингу 3.12.

Лістинг 3.12 – Функція для завантаження даних з файлу

```
def load_from_file(filename):
    with open(filename, 'rb') as f:
        return f.read()
```

Функції save_key_to_file(key, filename), load_key_from_file(filename), read_plaintext(input_file), save_to_file(data, filename) та load_from_file(filename) можна назвати допоміжними функціями, оскільки вони потрібні для роботи з файлами, а саме зчитування повідомлення або ключів. Однак, без них робота

програного засобу практично не можлива, оскільки застосунок працює саме з файлами.

Основні функції, без яких застосунок не зможе виконувати своєї задачі, будуть реалізовані в даному блоці.

Першою буде функція encrypt_AES_GCM(msg, secretKey), яка шифрує повідомлення за допомогою алгоритму AES у режимі GCM (Galois/Counter Mode). Приймає повідомлення у вигляді байтів та секретний ключ, повертає зашифроване повідомлення, вектор ініціалізації (nonce) та аутентифікаційний тег. Реалізація даної функції наведена в лістингу 3.13.

Лістинг 3.13 – Функція для шифрування повідомлення на основі AES

```
def encrypt_AES_GCM(msg, secretKey):
    aesCipher = AES.new(secretKey, AES.MODE_GCM)
    ciphertext, authTag = aesCipher.encrypt_and_digest(msg)
    return ciphertext, aesCipher.nonce, authTag
```

Функція decrypt_AES_GCM(ciphertext, nonce, authTag, secretKey) розшифровує повідомлення, зашифроване за допомогою AES-GCM. Приймає зашифроване повідомлення, вектор ініціалізації, аутентифікаційний тег та секретний ключ. Повертає розшифроване повідомлення у вигляді байтів. Реалізація даної функції наведена в лістингу 3.14.

Лістинг 3.14 – Функція для розшифрування повідомлення на основі AES

```
def decrypt_AES_GCM(ciphertext, nonce, authTag, secretKey):
    aesCipher = AES.new(secretKey, AES.MODE_GCM, nonce)
    plaintext = aesCipher.decrypt_and_verify(ciphertext,
authTag)
    return plaintext
```

Функція ecc_point_to_256_bit_key(point) перетворює точку еліптичної кривої в 256-бітний ключ. Використовує хешування координат точки (x та y) для отримання 256-бітного ключа у вигляді байтів. Реалізація даної функції наведена в лістингу 3.15.

Лістинг 3.15 – Функція для перетворення точки на ключ

```
def ecc_point_to_256_bit_key(point):
```

```
        sha = hashlib.sha256(int.to_bytes(point.x, 32, 'big'))
        sha.update(int.to_bytes(point.y, 32, 'big'))
        return sha.digest()
```

Функція encrypt_ECC(msg, pubKey) шифрує повідомлення за допомогою еліптичної кривої. Приймає повідомлення у вигляді байтів та публічний ключ (точку на кривій), генерує випадковий приватний ключ для сеансу, обчислює спільний секрет, який використовується для шифрування повідомлення за допомогою AES-GCM. Повертає зашифроване повідомлення, публічний ключ для зашифрованого повідомлення, вектор ініціалізації та аутентифікаційний тег. Реалізація даної функції наведена в лістингу 3.16[6].

Лістинг 3.16 – Функція для шифрування повідомлення за допомогою еліптичної кривої

```
    def encrypt_ECC(msg, pubKey):
        ciphertextPrivKey = secrets.randbelow(curve.field.n)
        sharedECCKey = ciphertextPrivKey * pubKey
        secretKey = ecc_point_to_256_bit_key(sharedECCKey)
        ciphertext, nonce, authTag = encrypt_AES_GCM(msg,
secretKey)
        ciphertextPubKey = ciphertextPrivKey * curve.g
        return ciphertext, ciphertextPubKey, nonce, authTag
```

Функція decrypt_ECC(encryptedMsg, privKey) розшифровує зашифроване повідомлення за допомогою еліптичної кривої. Приймає зашифроване повідомлення (яке містить зашифровані дані, публічний ключ, вектор ініціалізації та автентифікаційний тег) та приватний ключ. Обчислює спільний секретний ключ, використовує його для розшифровування повідомлення за допомогою AES-GCM. Повертає розшифроване повідомлення. Реалізація даної функції наведена в лістингу 3.17.

Лістинг 3.17 – Функція для розшифрування повідомлення за допомогою еліптичної кривої

```
    def decrypt_ECC(encryptedMsg, privKey):
        ciphertext, ciphertextPubKey, nonce, authTag =
encryptedMsg
```

```
        sharedECCKey = privKey * ciphertextPubKey
        secretKey = ecc_point_to_256_bit_key(sharedECCKey)
        plaintext = decrypt_AES_GCM(ciphertext, nonce, authTag,
secretKey)
        return plaintext
```

Функція generate_keys() генерує та зберігає пару ключів (приватний та публічний) у відповідні файли. Створює випадковий приватний ключ, обчислює відповідний публічний ключ і зберігає їх у файли private_key.pem, public_key_x.pem та public_key_y.pem. Реалізація даної функції наведена в лістингу 3.18.

Лістинг 3.18 – Функція для генерації ключів на еліптичній кривій

```
def generate_keys():
    privKey = secrets.randbelow(curve.field.n)
    pubKey = privKey * curve.g
    save_key_to_file(privKey, 'private_key.pem')
    save_key_to_file(pubKey.x, 'public_key_x.pem')
    save_key_to_file(pubKey.y, 'public_key_y.pem')
```

Функція select_keys() вибирає файли з ключами через діалогове вікно. Відкриває файлові діалоги для вибору файлів приватного ключа та координат публічного ключа (x і y), показує повідомлення про успішний вибір ключів або попередження, якщо не всі файли вибрані. Реалізація даної функції наведена в лістингу 3.19.

Лістинг 3.19 – Функція для вибору ключів

```
def select_keys():
    global privKey_file, pubKey_x_file, pubKey_y_file
    privKey_file = filedialog.askopenfilename(title="Select
private key file")
    pubKey_x_file = filedialog.askopenfilename(title="Select
public key x file")
    pubKey_y_file = filedialog.askopenfilename(title="Select
public key y file")
    if privKey_file and pubKey_x_file and pubKey_y_file:
```

```
            messagebox.showinfo("Success", "Keys selected
successfully.")
        else:
            messagebox.showwarning("Warning", "Please select all
key files.")
```

Функція encrypt_file() шифрує вміст файлу за допомогою еліптичної кривої та AES-GCM. Зчитує вміст вибраного файлу, зашифровує його публічним ключем, зберігає зашифрований файл з розширенням .encrypted у вибраній директорії та виводить повідомлення про успішне шифрування. Реалізація даної функції наведена в лістингу 3.20.

Лістинг 3.20 – Функція для шифрування вмісту файлу

```
    def encrypt_file():
        plaintext_file =
filedialog.askopenfilename(title="Select a file to encrypt")
        if plaintext_file and pubKey_x_file and pubKey_y_file:
            msg = read_plaintext(plaintext_file)
            pubKey_x = load_key_from_file(pubKey_x_file)
            pubKey_y = load_key_from_file(pubKey_y_file)
            pubKey = ec.Point(curve, pubKey_x, pubKey_y)
            encryptedMsg = encrypt_ECC(msg, pubKey)
            save_dir = filedialog.askdirectory(title="Select
directory to save encrypted file")
            if save_dir:
                encrypted_file_path = os.path.join(save_dir,
os.path.basename(plaintext_file) + '.encrypted')
                with open(encrypted_file_path, 'wb') as f:
                    f.write(os.path.splitext(os.path.basename(pl
aintext_file))[1].encode() + b'\n')
                    f.write(encryptedMsg[0])
                    f.write(encryptedMsg[1].x.to_bytes(32,
'big'))
                    f.write(encryptedMsg[1].y.to_bytes(32,
'big'))
                    f.write(encryptedMsg[2])
```

```
                    f.write(encryptedMsg[3])
                messagebox.showinfo("Success", f"File encrypted
and saved as '{encrypted_file_path}'.")
        else:
            messagebox.showwarning("Warning", "Please select a
file and keys first.")
```

Функція decrypt_file() розшифровує вміст зашифрованого файлу за допомогою приватного ключа. Зчитує зашифрований файл, розшифровує його за допомогою приватного ключа, зберігає розшифрований файл у вибраній директорії та виводить повідомлення про успішне розшифрування. Реалізація даної функції наведена в лістингу 3.21.

Лістинг 3.21 – Функція для розшифрування зашифрованого файлу

```
    def decrypt_file():
        encrypted_file =
filedialog.askopenfilename(title="Select a file to decrypt")
        if encrypted_file and privKey_file:
            encrypted_data = load_from_file(encrypted_file)
            original_extension =
encrypted_data.split(b'\n')[0].decode()
            encrypted_data =
encrypted_data[len(original_extension) + 1:]
            ciphertext = encrypted_data[:-96]
            pubKey_x = int.from_bytes(encrypted_data[-96:-64],
'big')
            pubKey_y = int.from_bytes(encrypted_data[-64:-32],
'big')
            nonce = encrypted_data[-32:-16]
            authTag = encrypted_data[-16:]
            ciphertextPubKey = ec.Point(curve, pubKey_x,
pubKey_y)
            encryptedMsg = (ciphertext, ciphertextPubKey, nonce,
authTag)
            privKey = load_key_from_file(privKey_file)
            decryptedMsg = decrypt_ECC(encryptedMsg, privKey)
```

```
            save_dir = filedialog.askdirectory(title="Select
directory to save decrypted file")
            if save_dir:
                decrypted_file_path = os.path.join(save_dir,
os.path.basename(encrypted_file).replace('.encrypted',
original_extension))
                with open(decrypted_file_path, 'wb') as f:
                    f.write(decryptedMsg)
                messagebox.showinfo("Success", f"File decrypted
and saved as '{decrypted_file_path}'.")
        else:
            messagebox.showwarning("Warning", "Please select a
file and keys first.")
```

Функція encrypt_text() шифрує текстові дані, введені користувачем, за допомогою публічного ключа. Зберігає введений текст у файл, зчитує його, шифрує за допомогою еліптичної кривої та AES-GCM, зберігає зашифрований текст у файл з розширенням .encrypted та виводить повідомлення про успішне шифрування. Реалізація даної функції наведена в лістингу 3.22.

Лістинг 3.22 – Функція для шифрування тексту з текстового поля

```
    def encrypt_text():
        if pubKey_x_file and pubKey_y_file:
            save_path =
filedialog.asksaveasfilename(defaultextension=".txt",
filetypes=[("Text files", "*.txt")])
            if save_path:
                msg = text_input.get("1.0", tk.END).encode()
                with open(save_path, 'wb') as f:
                    f.write(msg)
                with open(save_path, 'rb') as f:
                    msg = f.read()
                pubKey_x = load_key_from_file(pubKey_x_file)
                pubKey_y = load_key_from_file(pubKey_y_file)
                pubKey = ec.Point(curve, pubKey_x, pubKey_y)
                encryptedMsg = encrypt_ECC(msg, pubKey)
```

```
                encrypted_text_path = save_path + ".encrypted"
                with open(encrypted_text_path, 'wb') as f:
                    f.write(encryptedMsg[0])
                    f.write(encryptedMsg[1].x.to_bytes(32,
'big'))
                    f.write(encryptedMsg[1].y.to_bytes(32,
'big'))
                    f.write(encryptedMsg[2]
                    f.write(encryptedMsg[3])
                messagebox.showinfo("Success", "Text encrypted
and saved successfully.")
        else:
            messagebox.showwarning("Warning", "Please select
keys first.")
```

Останньою, але не менш важливою є функція save_keys(), яка зберігає згенеровані ключі у відповідні файли. Вибирає директорію для збереження ключів, генерує випадковий приватний ключ, обчислює відповідний публічний ключ, зберігає ключі у файли private_key.pem, public_key_x.pem та public_key_y.pem та виводить повідомлення про успішне збереження ключів. Реалізація даної функції наведена в лістингу 3.23.

Лістинг 3.23 – Функція для збереження ключів у файли

```
    def save_keys():
        global privKey_file, pubKey_x_file, pubKey_y_file
        save_dir = filedialog.askdirectory(title="Select
directory to save keys")
        if save_dir:
            privKey_file = os.path.join(save_dir,
'private_key.pem')
            pubKey_x_file = os.path.join(save_dir,
'public_key_x.pem')
            pubKey_y_file = os.path.join(save_dir,
'public_key_y.pem')
            privKey = secrets.randbelow(curve.field.n)
            pubKey = privKey * curve.g
```

```
            save_key_to_file(privKey, privKey_file)
            save_key_to_file(pubKey.x, pubKey_x_file)
            save_key_to_file(pubKey.y, pubKey_y_file)

            messagebox.showinfo("Success", "Keys generated and
saved.")
```

На даному етапі вже реалізовані всі основні та допоміжні функції, які дозволяють використовувати весь функціонал, який був визначений.

3.3 Тестування програмного забезпечення

Тестування програмного забезпечення є одним з найважливіших етапів розробки програмного продукту. Це процес, який передбачає перевірку програмного коду з метою виявлення помилок та забезпечення відповідності продукту заданим вимогам. Без належного тестування якість програмного забезпечення може значно постраждати, що призведе до незадоволеності користувачів та додаткових витрат на виправлення помилок після релізу.

Основні цілі тестування:

– Виявлення дефектів. Основна мета тестування полягає у виявленні дефектів (помилок) у програмному забезпеченні. Тестувальники перевіряють кожен аспект програми, щоб знайти будь-які невідповідності між фактичною поведінкою програми та очікуваною. Це допомагає зменшити кількість багів, які можуть виникнути в кінцевому продукті.

– Забезпечення якості. Тестування допомагає забезпечити високу якість програмного продукту. Висока якість означає, що продукт відповідає усім заданим вимогам, працює стабільно та не має критичних помилок. Це також включає відповідність стандартам безпеки, продуктивності та зручності використання.

– Перевірка відповідності вимогам. Перед початком розробки програмного забезпечення складаються вимоги, які описують, як продукт повинен працювати. Тестування допомагає переконатися, що кінцевий

продукт відповідає цим вимогам, включаючи функціональні та нефункціональні аспекти.

– Підвищення задоволеності користувачів. Якісне програмне забезпечення підвищує задоволеність користувачів. Якщо продукт працює без помилок, має зрозумілий інтерфейс та виконує свої функції належним чином, користувачі будуть більше задоволені, що сприятиме зростанню довіри до компанії та підвищенню її репутації.

Види тестування

– Функціональне тестування. Цей вид тестування перевіряє, чи виконує програмне забезпечення всі свої функції відповідно до вимог. Це включає тестування окремих функцій, інтеграцію між модулями та перевірку кінцевого результату.

– Нефункціональне тестування. Нефункціональне тестування охоплює аспекти, що не стосуються конкретних функцій програми, такі як продуктивність, масштабованість, безпека та зручність використання. Це тестування допомагає визначити, наскільки добре продукт працює під різними навантаженнями, в різних умовах та наскільки він захищений від потенційних загроз.

Дане тестування направлене на перевірку функціональних вимог до програмного застосунку. Тестування проводилося на операційних системах Windows 10, 11. Даний застосунок займає всього 180Кб, не використовує багато оперативної пам’яті комп’ютера та дозволяє працювати іншим програмам без труднощів.

В процесі тестування було визначено, що кнопки користувацького інтерфейсу мають достатній розмір, розташовані в відповідно названих блоках. Текст достатнього розміру, читабельний та дозволяє зручно використовувати програму. Кнопки підписані відповідно до їх функцій, англійською мовою.

Тестуванню також підлягають наступні аспекти програми:

– Тестування генерування ключів.

– Тестування вибору ключів.

– Тестування шифрування файлу.

– Тестування шифрування введеного тексту.

– Тестування розшифрування файлу.

Проведемо тестування програмного засобу згідно тестового плану.

Для шифрування файлів чи введеного тексту потрібні ключі, які має генерувати програма. Щоб їх згенерувати користувач тисне на кнопку «Generate key» у блоці «Key Management», як зображено на рисунку 3.2.

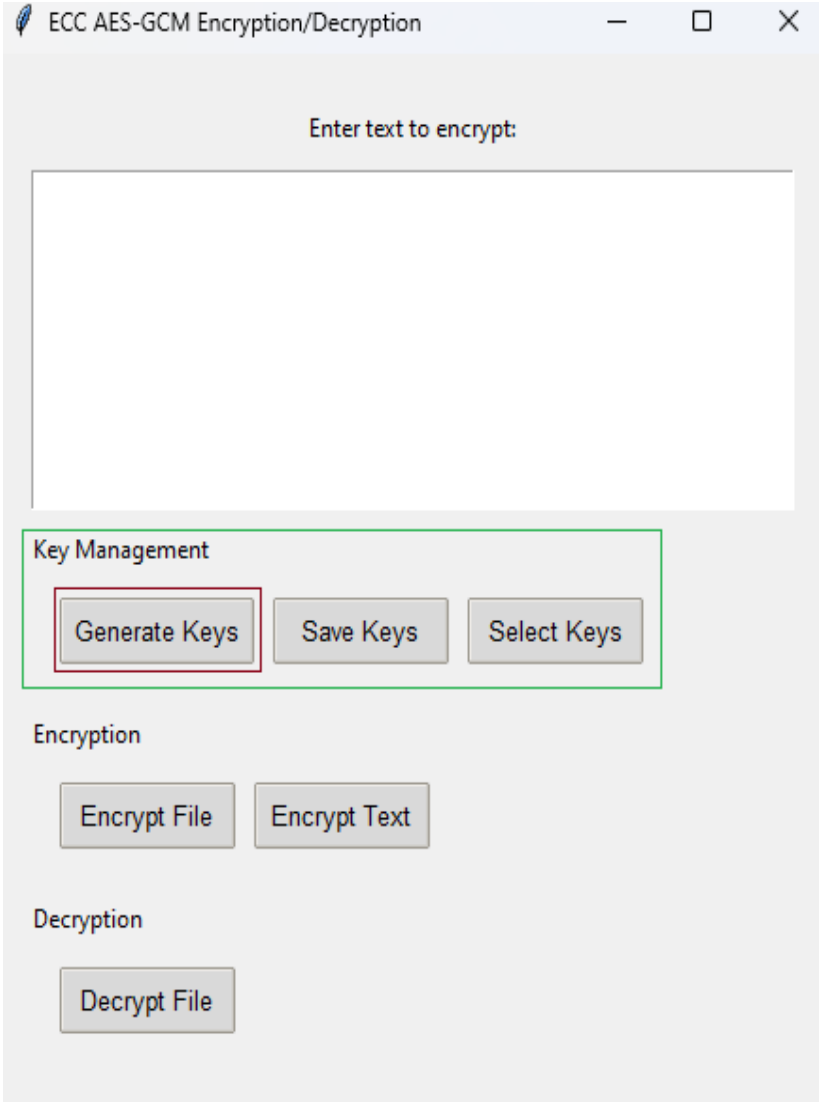

Рисунок 3.2 – Кнопка для генерації ключів

В результаті будуть згенеровані ключі. Однак, для подальшого використання програми їх потрібно спочатку зберегти у файли. Для цього використовуємо кнопку «Save Keys» також у блоці «Key Management», що зображено на рисунку 3.3.

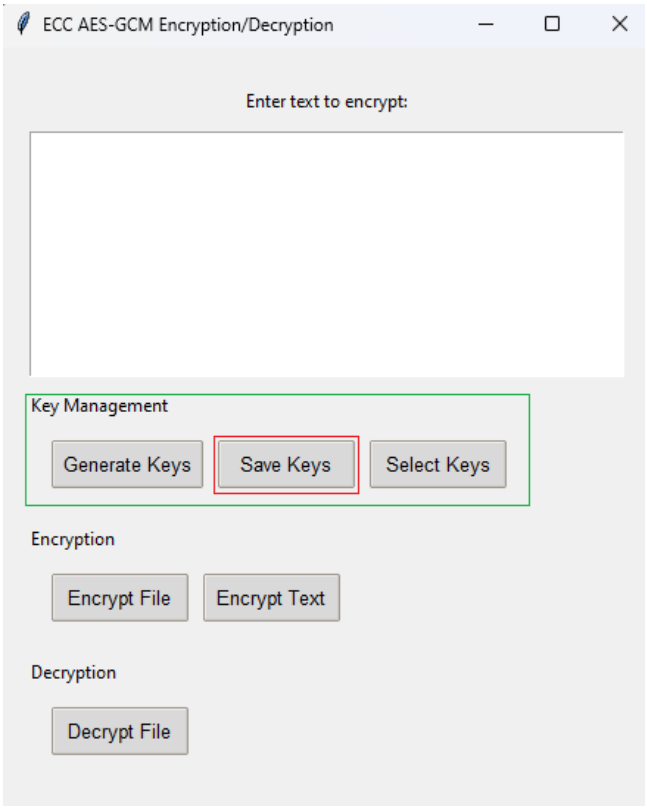

Рисунок 3.3 – Кнопка для збереження ключів

Тоді перед користувачем відкривається вікно для вибору місця, куди ці ключі будуть збережені. Це продемонстровано на рисунку 3.4.

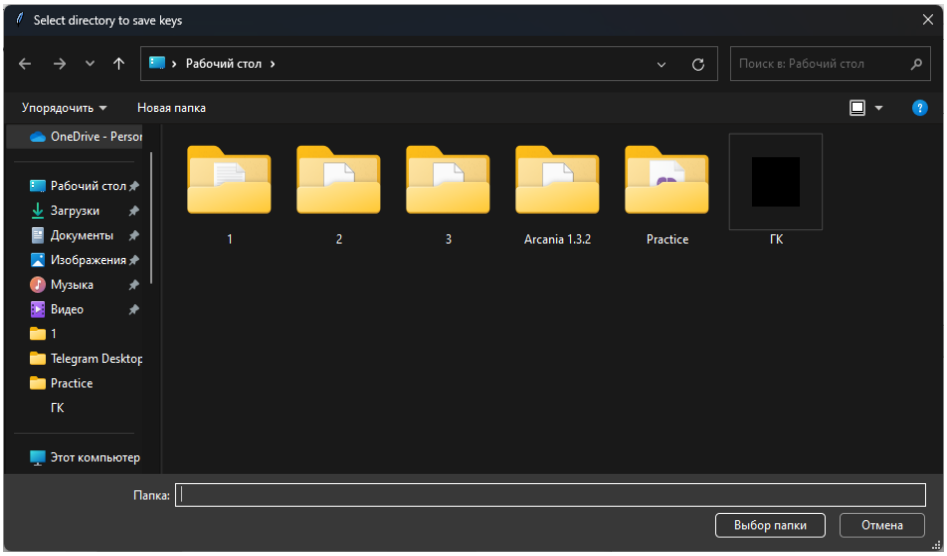

Рисунок 3.4 – Вікно вибору шляху збереження

Після цього з’явиться повідомлення, що ключі успішно збережені, як на рисунку 3.5.

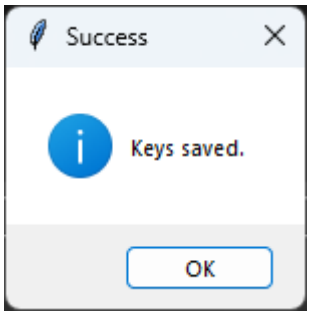

Рисунок 3.5 – Повідомлення про збереження ключів у файли

Тепер все готово для шифрування файлу або введеного тексту. Спершу протестуємо шифрування готових файлів. Для цього користувач тисне на кнопку «Encrypt File», яка розташована у блоці «Encryption», як зображено на рисунку 3.6.

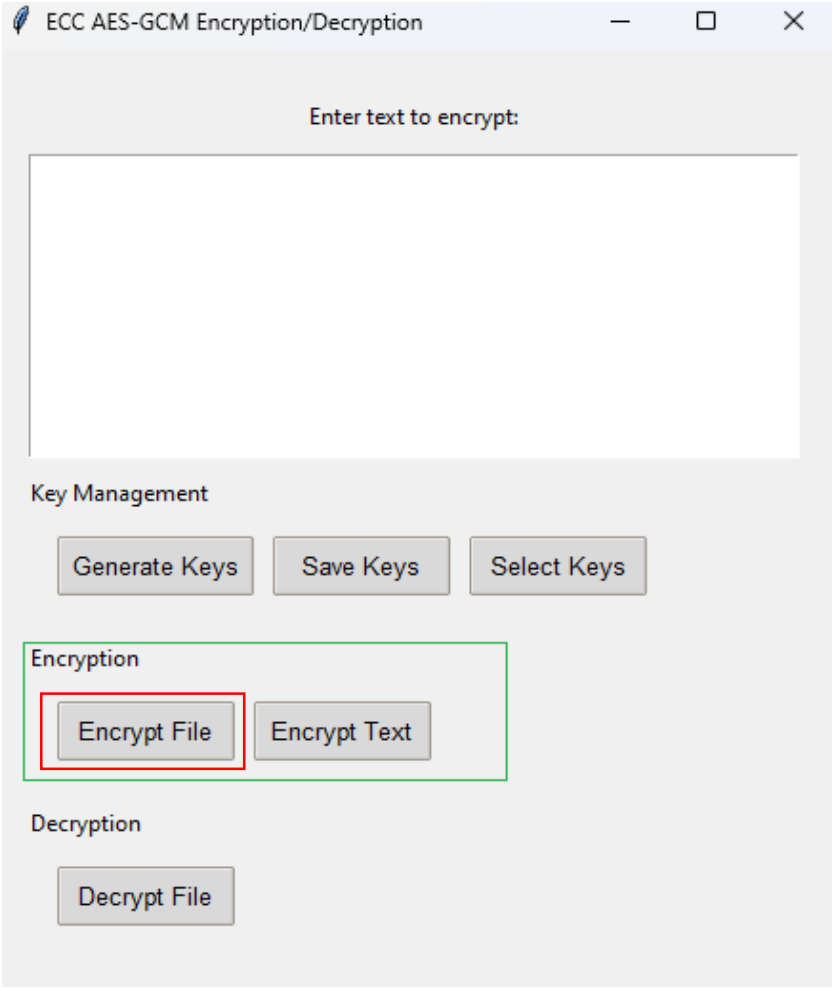

Рисунок 3.6 – Кнопка для шифрування файлу

Після натиснення на кнопку «Encrypt File» відкриється вікно вибору файлу для шифрування, яке зображене на рисунку 3.7.

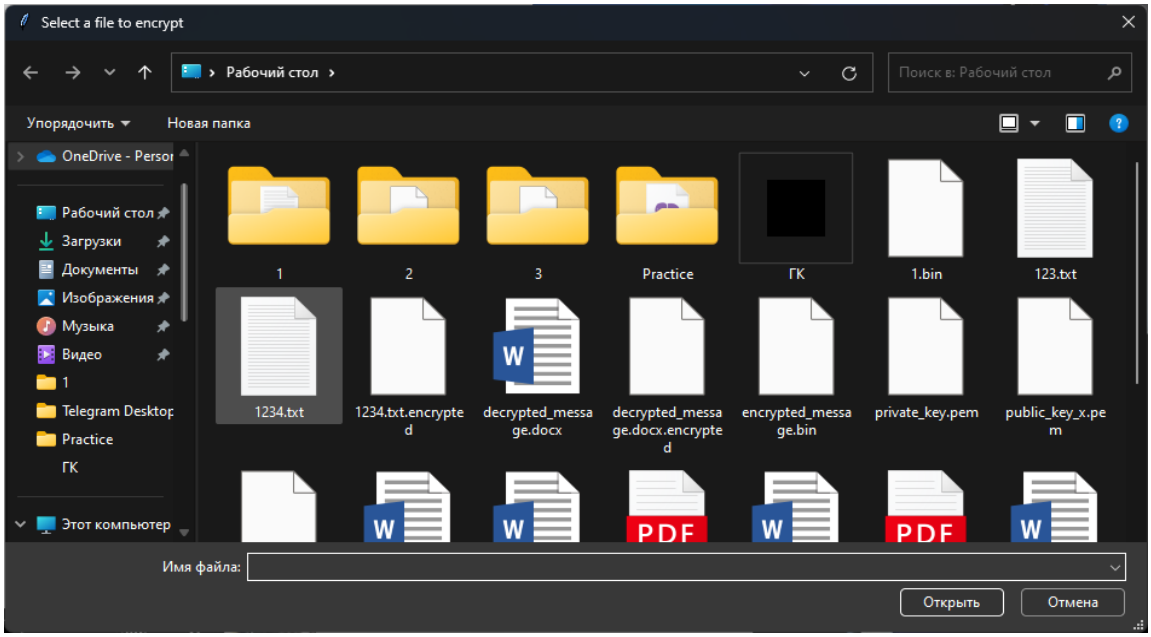

Рисунок 3.7 – Вікно вибору файлу для шифрування

Коли користувач обрав файл, який бажає зашифрувати, одразу відкривається таке ж вікно, яке пропонує вибрати шлях, де куди буде збережений зашифрований файл. В результаті перед користувачем з’явиться повідомлення про успішне шифрування файлу та збереження його до вказаного шляху. Вміст такого повідомлення продемонстрований на рисунку 3.8.

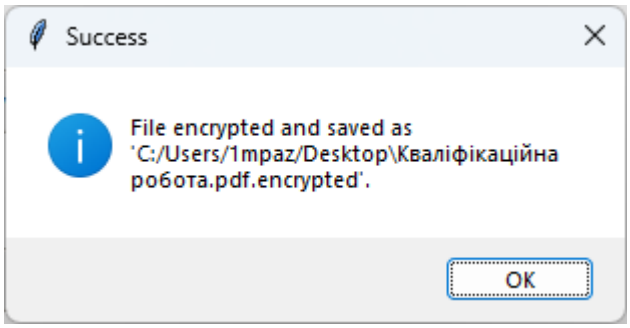

Рисунок 3.8 – Повідомлення про успішне шифрування файлу

Для шифрування введеного тексту користувачем порядок дій схожий. Після введення бажаного тексту, спочатку потрібно згенерувати ключі, як продемонстровано на рисунку 3.2 та зберегти їх у файли, як на рисунках 3.3 і 3.4. Однак після цих дій потрібно натиснути на кнопку «Encrypt File». В

результаті відкриється вікно для вибору шляху куди буде зберігатися зашифрований текст. Тут потрібно вказати назву для файлу, куди буде збережений зашифрований текст. Це зображено на рисунку 3.9.

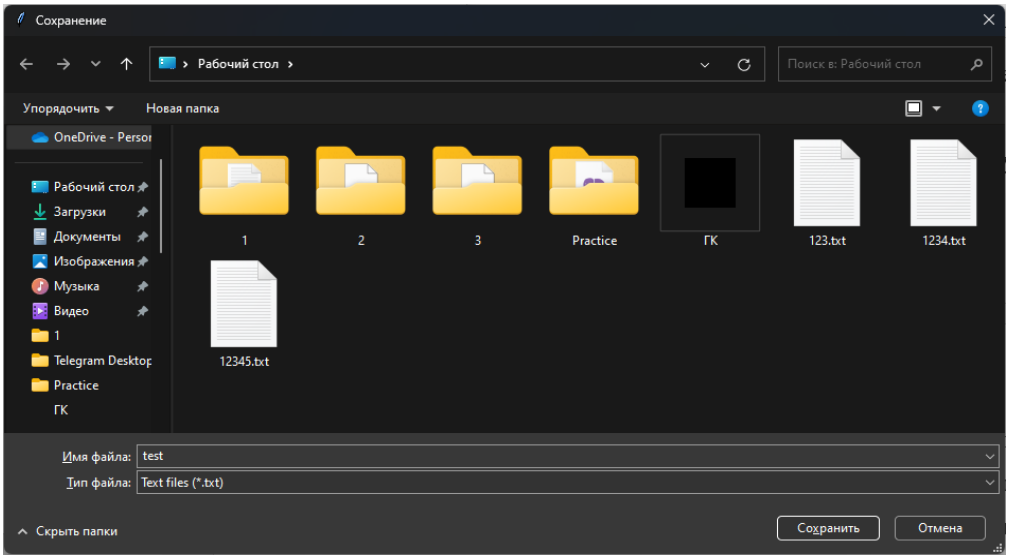

Рисунок 3.9 – Вікно вибору шляху збереження тексту

Як результат, отримуємо текстовий файл з введеним текстом та файл з таким же іменем, однак його розширення .encrypted. Таке розширення використовують зашифровані файли.

Для перевірки правильності шифрування можна відкрити один із зашифрованих файлів та переглянути їх вміст, який зображений на рисунку 3.10.

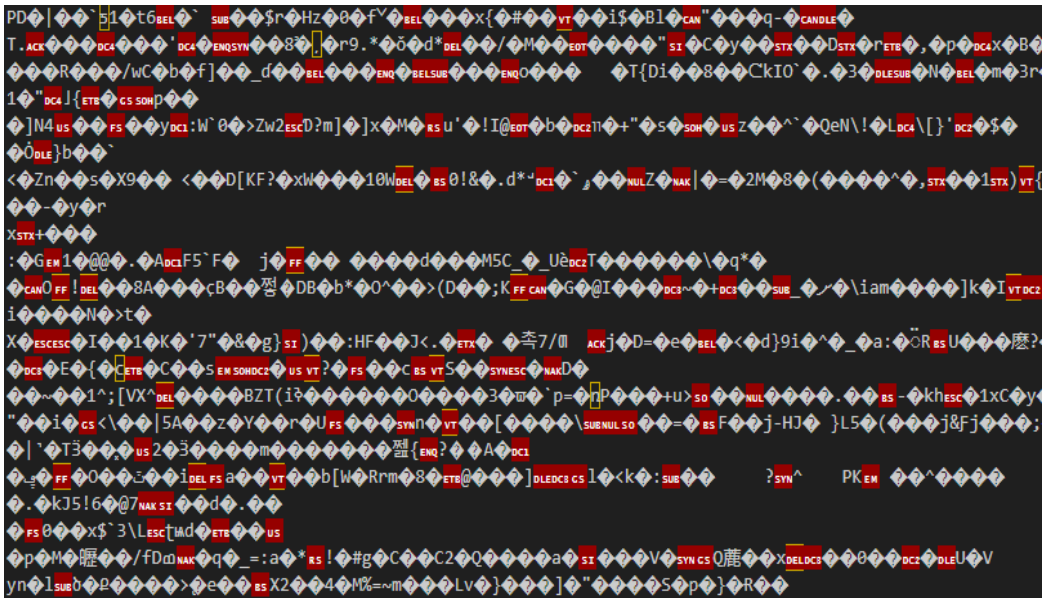

Рисунок 3.10 – Вигляд зашифрованого вмісту файлу

Тепер можна переходити до тестування розшифрування зашифрованого файлу.

Для розшифрування файлу потрібні ключі, якими був зашифрований файл. Перед початком розшифрування їх потрібно вибрати, щоб програма могла з ними працювати. Тому спочатку треба обрати ключі. Для цього потрібно натиснути на кнопку «Select Keys», шо розташована в блоці «Key Management». Дана кнопка зображена на рисунку 3.11.

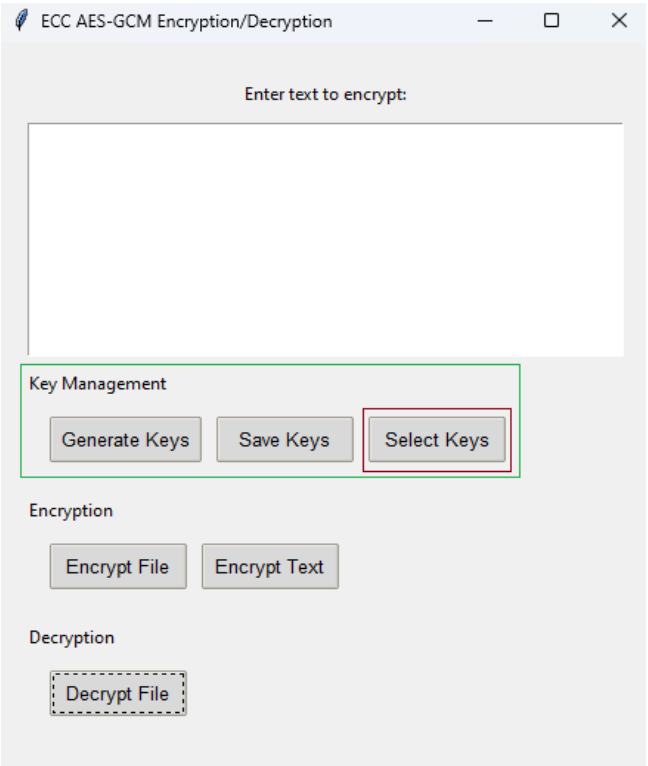

Рисунок 3.11 – Кнопка для вибору ключів

Після натиснення на неї відкриється вікно для вибору ключів. Однак, важливо пам’ятати, що ключі зберігаються в 3 файлах. Зберігається приватний ключ, а також, точки x та y публічного ключа, тому їх порядок вибору важливий. Спочатку треба обрати приватний ключ, слідом файл з точкою x, і в кінці файл з точкою y.

Тому спочатку обираємо приватний ключ, як на рисунку 3.12.

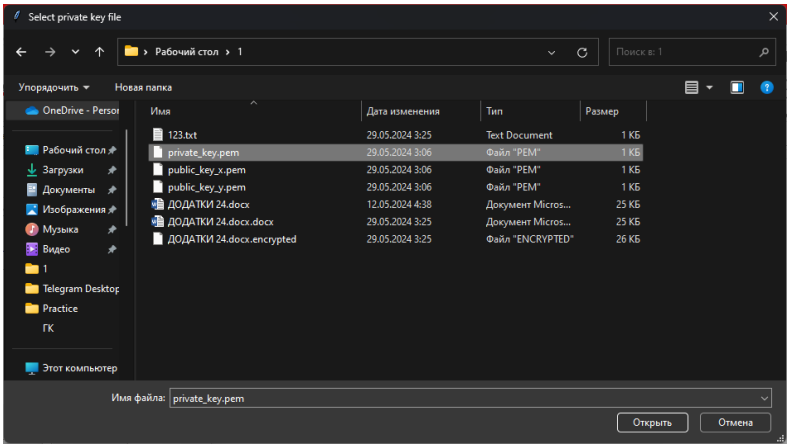

Рисунок 3.12 – Вибір приватного ключа

Далі обираємо точку x публічного ключа користувача, як на рисунку 3.13.

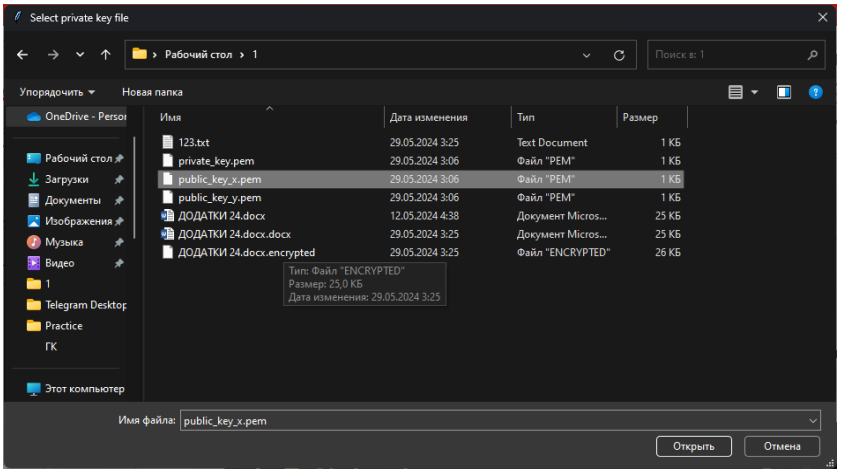

Рисунок 3.13 – Вибір точки x публічного ключа

І останнім обираємо точку y публічного ключа, як на рисунку 3.14.

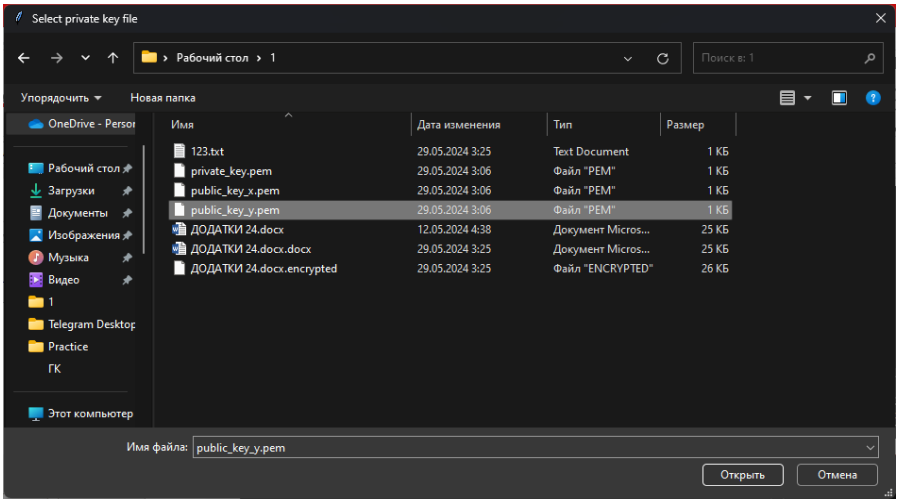

Рисунок 3.14 – Вибір точки y публічного ключа

Після того, як були обрані ключі, з’явиться повідомлення, яке зображене на рисунку 3.15.

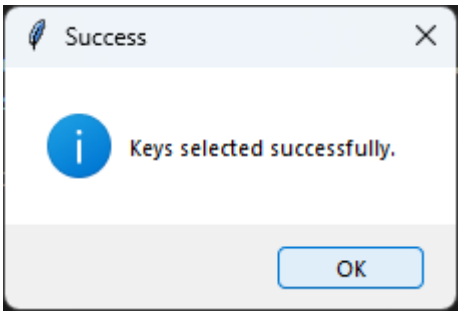

Рисунок 3.15 – Повідомлення про успішний вибір ключів

Тепер можна розшифровувати файл. Для цього потрібно натиснути на кнопку «Decrypt File», що розташована у блоці «Decryption». Дана кнопка зображена на рисунку 3.16.

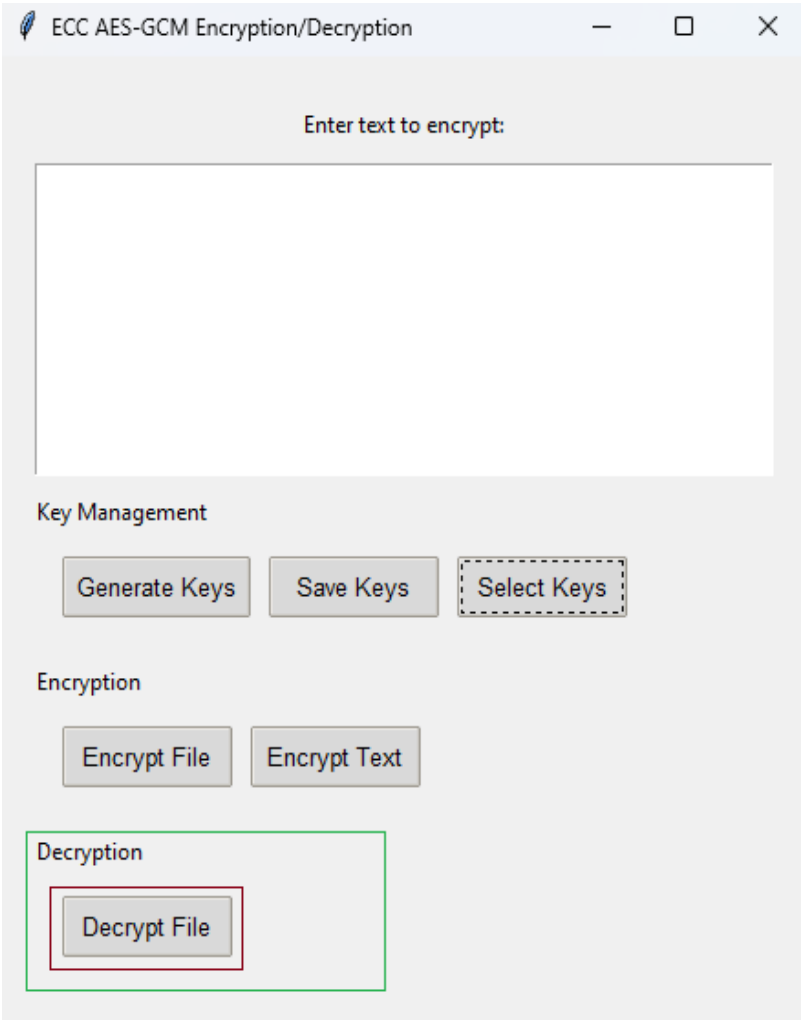

Рисунок 3.16 – Кнопка для розшифрування файлу

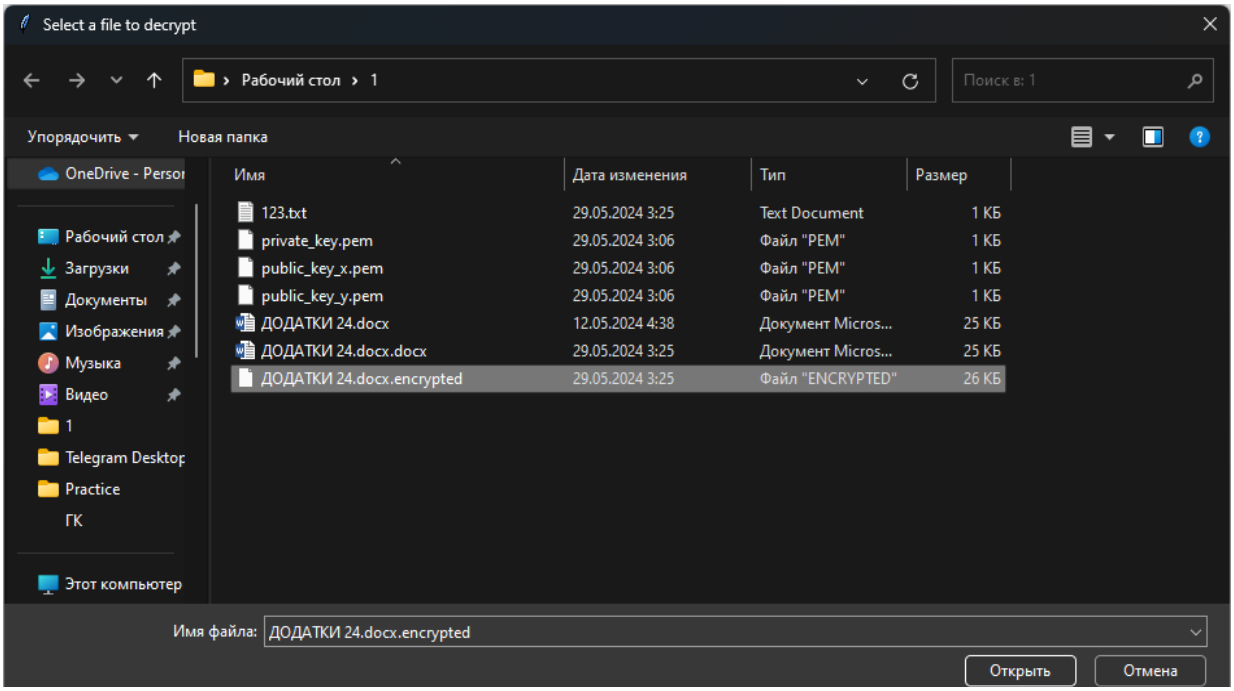

Рисунок 3.17 – Вікно вибору зашифрованого файлу

В результаті відкриється вікно для вибору зашифрованого файлу, як на рисунку 3.17.

Після того, як був обраний файл, знову відкриється вікно, де потрібно обрати куди зберегти розшифрований файл. Коли файл буде розшифрований і збережений за вказаним шляхом, з’явиться повідомлення, вміст якого зображений на рисунку 3. 18.

Тепер за вказаним шляхом буде знаходитися розшифрований файл, вміст якого повністю відповідає файлу перед шифруванням.

Отже тестування програмного забезпечення є невід'ємною частиною процесу розробки. Воно забезпечує високу якість продукту, задоволеність користувачів та відповідність вимогам.

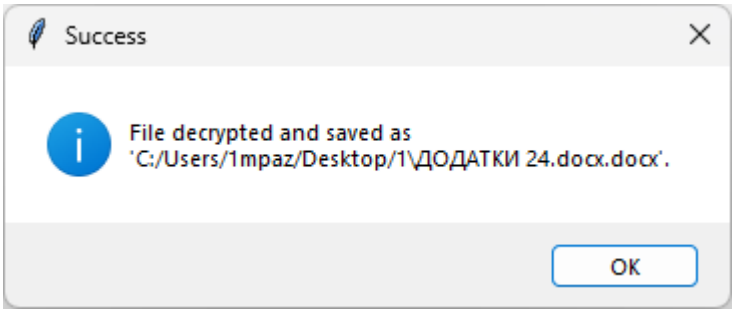

Рисунок 3.18 – Повідомлення про успішне розшифрування файлів

Проведення тестування програмного забезпечення є важливим етапом у процесі його розробки. Тестування дозволяє переконатися, що продукт відповідає вимогам, працює стабільно та є безпечним. Процес тестування має певні критерії та особливості, які потребують уваги.

Тестування було пройдене згідно плану, був перевірений користувацький інтерфейс та основні функції програми. Програма відповідає поставленим вимогам, функції працюють правильно, без видимих проблем. Даний додаток є зручним у використанні, навіть не дуже впевнений користувачі зможе з легкістю його використовувати.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Програмний засіб асиметричного шифрування файлів на основі еліптичних кривих (ECC) є сучасним і ефективним рішенням для захисту даних. Основні технічні характеристики включають використання ECC для забезпечення високої криптографічної стійкості при менших розмірах ключів, що знижує вимоги до обчислювальних ресурсів. Шифрування на основі ECC дозволяє отримати еквівалентний рівень безпеки за менший розмір ключа у порівнянні з RSA (наприклад, ключ ECC розміром 256 біт забезпечує безпеку, еквівалентну ключу RSA розміром 3072 біт).

З економічної точки зору, зменшення розмірів ключів і підвищення ефективності обробки даних дозволяє знизити витрати на апаратне забезпечення та споживання енергії. Крім того, програмний засіб легко інтегрується в існуючі системи безпеки, що знижує витрати на впровадження і підтримку.

Даний програмний засіб є новим на ринку, оскільки він використовує передові алгоритми ECC, які до цього часу не були широко впроваджені у багатьох галузях. Однак, деякі елементи можуть бути модифікаціями вже існуючих рішень, що використовують еліптичні криві для шифрування даних. Унікальні особливості нового виробу включають оптимізацію алгоритмів для різних типів файлів та покращену інтеграцію з популярними платформами.

Основними потенційними замовниками є компанії та організації, які мають високі вимоги до безпеки даних. Це можуть бути фінансові установи, урядові організації, медичні заклади, а також великі корпорації, що працюють з конфіденційною інформацією. Крім того, програмний засіб буде цікавий для розробників програмного забезпечення, які бажають інтегрувати функції безпечного шифрування у свої продукти.

Очікується високий попит на даний виріб, враховуючи зростання кількості загроз і збільшення обсягу конфіденційної інформації, що зберігається в цифровому форматі. Програмний засіб може задовольнити потреби як великих компаній, так і середніх підприємств, що шукають надійні та ефективні рішення для шифрування даних.

Для продажу програмного засобу слід віддати перевагу кільком методам, зокрема прямий продаж через корпоративні канали, використання партнерських мереж, а також онлайн-продажі через офіційний веб-сайт і платформи електронної комерції. Додатково можна використовувати маркетингові кампанії, спрямовані на підвищення обізнаності про продукт серед цільової аудиторії.

Можливі обсяги продажу залежать від активної маркетингової стратегії та зростання обізнаності про переваги ECC. Враховуючи зростання попиту на рішення для кібербезпеки, можна очікувати стабільне зростання продажів на рівні 15-20% щороку протягом перших п’яти років після виходу на ринок.

На ринку аналогічної продукції основними конкурентами є компанії, що пропонують рішення для шифрування даних, такі як RSA Security, Symantec (Norton), McAfee, а також нові гравці, які впроваджують технології на основі квантових обчислень.

4.2 Розрахунок витрат на розробку проєкту

Розробка програмного застосунку для шифрування файлів на основі еліптичних кривих (ECC) є складним та багатоетапним процесом, який вимагає значних ресурсів та детального планування. Цей проєкт включає в себе залучення висококваліфікованих фахівців, використання сучасних технологій та забезпечення надійної інфраструктури для розробки, тестування та підтримки програмного забезпечення. Враховуючи високу потребу в безпеці даних у сучасному світі, інвестиції в такий проєкт є стратегічно важливими для будь-якої організації, що прагне захистити свою інформацію.

Основними компонентами витрат на розробку програмного забезпечення є заробітні плати команди розробників, витрати на інфраструктуру, ліцензії на програмні продукти та інструменти для розробки, а також витрати на забезпечення офісного простору та обладнання. Кожен з цих елементів вимагає ретельного розрахунку для забезпечення ефективного використання ресурсів та досягнення поставлених цілей у встановлені терміни. Це відображено в таблиці 4.1.

Таблиця 4.1 – Кошторис витрат на створення проєкту

Призначення позицій витрат	Сума, грн	Пояснення
1. Заробітня плата програміста	20000	
2. Вирахування на соціальні потреби	4400	ЄСВ – 22%
3. Контрагентські роботи і послуги	2000	20000 – 10%
4. Витрати на відрядження	0	
5. Інші прямі витрати	3000	15%
6. Усього прямих витрат	29400	20000 + 4400 + 2000 + 3000
7. Накладні витрати	6174	29400– 21%
8. Планові накопичення	8893,5	(29400 + 6174) – 25%
9. Усього, кошторисна вартість проєкту	44467,5	29400+ 6174+ 8893
10. Податок на додану вартість	8893,5	9680 – 20%
11. Загалом, договірна ціна розробки ПЗ	53361	9680+1936

Заробітня плата розраховується на одного працівника, розробника, як це показано в таблиці 4.2.

Таблиця 4.2 - розрахунок платні

N	Посада	Оклад	Відрахування	Кількість		
п/п	виконавця	грн/міс	грн/міс	Чол.	Місяців	з/п, грн
1	Junior - developer	25000	5000	1	4	20000
2	Midle - developer	50000	10000	1	2	40000
3	Тестувальник	20000	4000	1	2	16000

4.3 Обгрунтування необхідності та розробки

Основними потребами замовників, які повинен задовольнити пропонований програмний засіб асиметричного шифрування файлів на основі еліптичних кривих, є забезпечення високого рівня безпеки даних, ефективний захист конфіденційної інформації від несанкціонованого доступу, відповідність нормативним вимогам щодо захисту даних, і зменшення ризиків, пов'язаних з витоками інформації. Також важливою потребою є підвищення ефективності та швидкості обробки даних завдяки використанню оптимізованих алгоритмів шифрування, що дозволяє зменшити навантаження на апаратне забезпечення і знизити витрати на обчислювальні ресурси.

Пропонований виріб вплине на поліпшення економічних показників замовників завдяки зменшенню витрат на апаратне забезпечення та енергоспоживання, підвищенню продуктивності та ефективності обробки даних, зниженню витрат на забезпечення відповідності нормативним вимогам, а також зменшенню втрат від можливих інцидентів з витоками інформації. Інтеграція з існуючими системами безпеки дозволить мінімізувати витрати на впровадження та підтримку нового рішення, що також сприятиме покращенню загальної економічної ефективності.

Основними напрямками отримання ефекту при запровадженні проєкту є підвищення рівня захисту даних, що зменшує ризики витоків і втрат

інформації, а також збільшення довіри клієнтів і партнерів до організації. Додаткові ефекти включають підвищення продуктивності та швидкості обробки даних, що дозволяє швидше і ефективніше виконувати операції, а також зменшення витрат на обчислювальні ресурси завдяки використанню більш ефективних алгоритмів шифрування. Крім того, інтеграція з існуючими системами безпеки зменшує витрати на впровадження та технічне обслуговування нового програмного засобу, що також позитивно впливає на загальні економічні показники.

ВИСНОВКИ

У кваліфікаційній роботі було проаналізовано предметну область та поставлено завдання розробити програмний засіб для шифрування файлів на основі еліптичних кривих. Під час постановки завдання були зазначені вимоги до програми, визначено її функціональні можливості. Були проаналізовані існуючі рішення на ринку.

Було проведене дослідження алгоритму ECC, як ефективного та широко розповсюдженого алгоритму шифрування даних. Проаналізована криптостійкість алгоритму ECC, а також сама еліптична крива та операції над її точками. Було досліджено створення приватного та публічного ключів для шифрування та алгоритм, на основі якого відбувається шифрування та розшифрування файлів.

Було проведено розробку та тестування програмного засобу для шифрування файлів на основі еліптичних кривих. Був спроєктований та створений користувацький інтерфейс засобу. Був реалізований весь функціонал програмного застосунку та проведено тестування для перевірки працездатності реалізований функцій та інтерфейсу.

Було виконано техніко-економічне обґрунтування в якому проводився аналіз ринку засобів шифрування та способи для реалізації власного застосунку. Був проведений розрахунок витрат на реалізацію програмного застосунку, обґрунтування необхідності в створенні такого програмного засобу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Еліптична крива. Вікіпедія : вебсайт. URL : https://uk.wikipedia.org/wiki/Еліптична_крива (дата звернення 14.12.2023)

2. Еліптична криптографія. Вікіпедія : вебсайт. URL : https://uk.wikipedia.org/wiki/Еліптична_криптографія (дата звернення 14.12.2023)

3. Що кожен розробник має знати про еліптичні криві. Medium : вебсайт. URL : https://medium.com/@saimon777/що-кожен-розробник-має-знати-про-еліптичні-криві-4a4adf03f3e6 (дата звернення 15.12.2023)

4. Practical-Cryptography-for-Developers-Book. GitHub : вебсайт. URL : https://github.com/nakov/Practical-Cryptography-for-Developers-Book (дата звернення 10.01.2024)

5. Welcome to PyCryptodome’s documentation. PyCryptodome : вебсайт. URL : https://pycryptodome.readthedocs.io/en/latest/ (дата звернення 11.01.2024)

6. Tinyec. GitHub : веб-сайт. URL : https://github.com/alexmgr/tinyec/tree/master (дата звернення 11.01.24)

7. Керівництво з використання Tkinter у Python: початок роботи та створення нового вікна. Друкарня веб-сайт. URL : https://drukarnia.com.ua/articles/kerivnictvo-z-vikoristannya-tkinter-u-python-pochatok-roboti-ta-stvorennya-novogo-vikna-T-A9- (дата звернення 23.04.2024)

8. Технології розробки програмних засобів асиметричного шифрування файлів на основі еліптичних кривих. Борак І. В. . ЗБІРНИК тез за матеріалами студентських науково-практичних конференцій в рамках Днів Науки 2024 в Галицькому фаховому коледжі імені В’ячеслава Чорновола (секція«Комп’ютерних технологій» та секція «Фізико-математичних та природничих дисциплін») – Тернопіль: Галицький фаховий коледж імені В’ячеслава Чорновола, 2024. ___с

ДОДАТКИ

Додаток А

Лістинг програмного засобу

```
import tkinter as tk
from tkinter import filedialog, messagebox
from tkinter import ttk
from tinyec import registry, ec
from Crypto.Cipher import AES
import hashlib, secrets
import os

#Генерація еліптичної кривої
curve = registry.get_curve('brainpoolP256r1')

def encrypt_AES_GCM(msg, secretKey):
    """
    Шифрує повідомлення за допомогою AES-GCM.

    Параметри:
        msg (bytes): Повідомлення для шифрування.
        secretKey (bytes): Секретний ключ для шифрування.

    Повертає:
        bytes: Зашифроване повідомлення.
        bytes: Вектор ініціалізації.
        bytes: Аутентифікаційний тег.
    """
    aesCipher = AES.new(secretKey, AES.MODE_GCM)
    ciphertext, authTag = aesCipher.encrypt_and_digest(msg)
    return ciphertext, aesCipher.nonce, authTag

def decrypt_AES_GCM(ciphertext, nonce, authTag, secretKey):
    """
```

```
        Розшифровує зашифроване повідомлення за допомогою AES-
GCM.

        Параметри:
            ciphertext (bytes): Зашифроване повідомлення.
            nonce (bytes): Вектор ініціалізації.
            authTag (bytes): Аутентифікаційний тег.
            secretKey (bytes): Секретний ключ для розшифрування.

        Повертає:
            bytes: Розшифроване повідомлення.
        """
        aesCipher = AES.new(secretKey, AES.MODE_GCM, nonce)
        plaintext = aesCipher.decrypt_and_verify(ciphertext,
authTag)
        return plaintext

    def ecc_point_to_256_bit_key(point):
        """
        Перетворює точку Еліптичної кривої в 256-бітний ключ.

        Параметри:
            point (ec.Point): Точка Еліптичної кривої.

        Повертає:
            bytes: 256-бітний ключ.
        """
        sha = hashlib.sha256(int.to_bytes(point.x, 32, 'big'))
        sha.update(int.to_bytes(point.y, 32, 'big'))
        return sha.digest()

    def encrypt_ECC(msg, pubKey):
        """
        Шифрує повідомлення за допомогою Еліптичної кривої.
```

```
        Параметри:
            msg (bytes): Повідомлення для шифрування.
            pubKey (ec.Point): Публічний ключ для шифрування.

        Повертає:
            bytes: Зашифроване повідомлення.
            ec.Point: Публічний ключ для зашифрованого
повідомлення.
            bytes: Вектор ініціалізації.
            bytes: Аутентифікаційний тег.
        """
        ciphertextPrivKey = secrets.randbelow(curve.field.n)
        sharedECCKey = ciphertextPrivKey * pubKey
        secretKey = ecc_point_to_256_bit_key(sharedECCKey)
        ciphertext, nonce, authTag = encrypt_AES_GCM(msg,
secretKey)
        ciphertextPubKey = ciphertextPrivKey * curve.g
        return ciphertext, ciphertextPubKey, nonce, authTag

    def decrypt_ECC(encryptedMsg, privKey):
        """
        Розшифровує зашифроване повідомлення за допомогою
Еліптичної кривої.

        Параметри:
            encryptedMsg (tuple): Кортеж із зашифрованим
повідомленням.
            privKey (int): Приватний ключ для розшифрування.

        Повертає:
            bytes: Розшифроване повідомлення.
        """
        ciphertext, ciphertextPubKey, nonce, authTag =
encryptedMsg
        sharedECCKey = privKey * ciphertextPubKey
```

```
        secretKey = ecc_point_to_256_bit_key(sharedECCKey)
        plaintext = decrypt_AES_GCM(ciphertext, nonce, authTag,
secretKey)
        return plaintext

    def save_key_to_file(key, filename):
        """
        Зберігає ключ у файл.

        Параметри:
            key (int): Ключ для зберігання.
            filename (str): Шлях до файлу, у який зберігається
ключ.
        """
        with open(filename, 'wb') as f:
            f.write(int.to_bytes(key, 32, 'big'))

    def load_key_from_file(filename):
        """
        Завантажує ключ з файлу.

        Параметри:
            filename (str): Шлях до файлу, з якого
завантажується ключ.

        Повертає:
            int: Завантажений ключ.
        """
        with open(filename, 'rb') as f:
            key = int.from_bytes(f.read(), 'big')
        return key

    def read_plaintext(input_file):
        """
        Зчитує текстовий файл.
```

```
        Параметри:
            input_file (str): Шлях до файлу для зчитування.

        Повертає:
            bytes: Вміст файлу.
        """
        with open(input_file, 'rb') as f:
            return f.read()

    def save_to_file(data, filename):
        """
        Зберігає дані у файл.

        Параметри:
            data (bytes): Дані для зберігання.
            filename (str): Шлях до файлу, у який зберігаються
дані.
        """
        with open(filename, 'wb') as f:
            f.write(data)

    def load_from_file(filename):
        """
        Завантажує дані з файлу.

        Параметри:
            filename (str): Шлях до файлу, з якого
завантажуються дані.

        Повертає:
            bytes: Завантажені дані.
        """
        with open(filename, 'rb') as f:
            return f.read()
```

```
    def generate_keys():
        """
        Генерує та зберігає пару ключів у відповідні файли.
        """
        privKey = secrets.randbelow(curve.field.n)
        pubKey = privKey * curve.g
        save_key_to_file(privKey, 'private_key.pem')
        save_key_to_file(pubKey.x, 'public_key_x.pem')
        save_key_to_file(pubKey.y, 'public_key_y.pem')

    def select_keys():
        """
        Вибирає файли з ключами.
        """
        global privKey_file, pubKey_x_file, pubKey_y_file
        privKey_file = filedialog.askopenfilename(title="Select
private key file")
        pubKey_x_file = filedialog.askopenfilename(title="Select
public key x file")
        pubKey_y_file = filedialog.askopenfilename(title="Select
public key y file")
        if privKey_file and pubKey_x_file and pubKey_y_file:
            messagebox.showinfo("Success", "Keys selected
successfully.")
        else:
            messagebox.showwarning("Warning", "Please select all
key files.")

    def encrypt_file():
        """
        Шифрує вміст файлу з використанням еліптичної кривої та
AES-GCM.

        Зчитує вміст файлу, зашифровує його публічним ключем,
```

```
        та зберігає зашифрований файл з розширенням
'.encrypted'.

        Вимоги:
            - Публічні ключі мають бути вибрані перед виконанням
функції.

        Виводить повідомлення про успішне шифрування.

        """
        plaintext_file =
filedialog.askopenfilename(title="Select a file to encrypt")
        if plaintext_file and pubKey_x_file and pubKey_y_file:
            msg = read_plaintext(plaintext_file)
            pubKey_x = load_key_from_file(pubKey_x_file)
            pubKey_y = load_key_from_file(pubKey_y_file)
            pubKey = ec.Point(curve, pubKey_x, pubKey_y)

            encryptedMsg = encrypt_ECC(msg, pubKey)

            save_dir = filedialog.askdirectory(title="Select
directory to save encrypted file")
            if save_dir:
                encrypted_file_path = os.path.join(save_dir,
os.path.basename(plaintext_file) + '.encrypted')
                with open(encrypted_file_path, 'wb') as f:
                    # Write original file extension
                    f.write(os.path.splitext(os.path.basename(pl
aintext_file))[1].encode() + b'\n')
                    f.write(encryptedMsg[0])  # Write ciphertext
                    f.write(encryptedMsg[1].x.to_bytes(32,
'big'))
                    f.write(encryptedMsg[1].y.to_bytes(32,
'big'))
                    f.write(encryptedMsg[2])  # Write nonce
```

```
                    f.write(encryptedMsg[3])  # Write authTag

                messagebox.showinfo("Success", f"File encrypted
and saved as '{encrypted_file_path}'.")
        else:
            messagebox.showwarning("Warning", "Please select a
file and keys first.")

    def decrypt_file():
        """
        Розшифровує вміст зашифрованого файлу.

        Зчитує та розшифровує вміст зашифрованого файлу,
        використовуючи приватний ключ.

        Вимоги:
            - Приватний ключ має бути вибраний перед виконанням
функції.

        Виводить повідомлення про успішне розшифрування.

        """
        encrypted_file =
filedialog.askopenfilename(title="Select a file to decrypt")
        if encrypted_file and privKey_file:
            encrypted_data = load_from_file(encrypted_file)
            original_extension =
encrypted_data.split(b'\n')[0].decode()
            encrypted_data =
encrypted_data[len(original_extension) + 1:]

            ciphertext = encrypted_data[:-96]
            pubKey_x = int.from_bytes(encrypted_data[-96:-64],
'big')
```

```
            pubKey_y = int.from_bytes(encrypted_data[-64:-32],
'big')
            nonce = encrypted_data[-32:-16]
            authTag = encrypted_data[-16:]

            ciphertextPubKey = ec.Point(curve, pubKey_x,
pubKey_y)
            encryptedMsg = (ciphertext, ciphertextPubKey, nonce,
authTag)

            privKey = load_key_from_file(privKey_file)

            decryptedMsg = decrypt_ECC(encryptedMsg, privKey)

            save_dir = filedialog.askdirectory(title="Select
directory to save decrypted file")
            if save_dir:
                decrypted_file_path = os.path.join(save_dir,
os.path.basename(encrypted_file).replace('.encrypted',
original_extension))
                with open(decrypted_file_path, 'wb') as f:
                    f.write(decryptedMsg)

                messagebox.showinfo("Success", f"File decrypted
and saved as '{decrypted_file_path}'.")
        else:
            messagebox.showwarning("Warning", "Please select a
file and keys first.")

    def encrypt_text():
        """
        Шифрує текстові дані, введені користувачем.

        Вимоги:
```

```
            - Публічні ключі мають бути вибрані перед виконанням
функції.

        Виводить повідомлення про успішне шифрування.

        """
        if pubKey_x_file and pubKey_y_file:
            # Вибір місця збереження тексту
            save_path =
filedialog.asksaveasfilename(defaultextension=".txt",
filetypes=[("Text files", "*.txt")])
            if save_path:
                msg = text_input.get("1.0", tk.END).encode()

                # Збереження тексту у вибраному місці
                with open(save_path, 'wb') as f:
                    f.write(msg)

                # Зчитування збереженого тексту
                with open(save_path, 'rb') as f:
                    msg = f.read()

                # Шифрування тексту
                pubKey_x = load_key_from_file(pubKey_x_file)
                pubKey_y = load_key_from_file(pubKey_y_file)
                pubKey = ec.Point(curve, pubKey_x, pubKey_y)

                encryptedMsg = encrypt_ECC(msg, pubKey)

                # Збереження зашифрованого тексту
                encrypted_text_path = save_path + ".encrypted"
                with open(encrypted_text_path, 'wb') as f:
                    f.write(encryptedMsg[0])  # Write ciphertext
                    f.write(encryptedMsg[1].x.to_bytes(32,
'big'))
```

```
                    f.write(encryptedMsg[1].y.to_bytes(32,
'big'))
                    f.write(encryptedMsg[2])  # Write nonce
                    f.write(encryptedMsg[3])  # Write authTag

                messagebox.showinfo("Success", "Text encrypted
and saved successfully.")
        else:
            messagebox.showwarning("Warning", "Please select
keys first.")

    def save_keys():
        """
        Зберігає згенеровані ключі у відповідні файли.

        Виводить повідомлення про успішне збереження ключів.

        """
        global privKey_file, pubKey_x_file, pubKey_y_file
        save_dir = filedialog.askdirectory(title="Select
directory to save keys")
        if save_dir:
            privKey_file = os.path.join(save_dir,
'private_key.pem')
            pubKey_x_file = os.path.join(save_dir,
'public_key_x.pem')
            pubKey_y_file = os.path.join(save_dir,
'public_key_y.pem')

            privKey = secrets.randbelow(curve.field.n)
            pubKey = privKey * curve.g
            save_key_to_file(privKey, privKey_file)
            save_key_to_file(pubKey.x, pubKey_x_file)
            save_key_to_file(pubKey.y, pubKey_y_file)
```

```
            messagebox.showinfo("Success", "Keys saved.")

    # Tkinter UI setup
    root = tk.Tk()
    root.title("ECC AES-GCM Encryption/Decryption")

    # Встановлення стилю ttk
    style = ttk.Style()
    style.theme_use("clam")  # Встановлюємо тему clam

    # Налаштовуємо кольори для класу TButton та шрифт
    style.configure('TButton', foreground='black',
background='#d9d9d9', font=('Arial', 10))

    frame = tk.Frame(root, padx=10, pady=10)
    frame.pack(padx=10, pady=10)

    text_input_label = tk.Label(frame, text="Enter text to
encrypt:")
    text_input_label.grid(row=0, column=0, columnspan=2, pady=5)

    text_input = tk.Text(frame, height=10, width=50)
    text_input.grid(row=1, column=0, columnspan=2, pady=5)

    # Key Management Frame
    key_frame = tk.LabelFrame(frame, text="Key Management",
padx=10, pady=10, borderwidth=0, highlightthickness=0)
    key_frame.grid(row=2, column=0, columnspan=2, pady=5,
sticky="ew")

    generate_keys_button = ttk.Button(key_frame, text="Generate
Keys", command=generate_keys)
    generate_keys_button.grid(row=0, column=0, pady=5, padx=5)
```

```
    save_keys_button = ttk.Button(key_frame, text="Save Keys",
command=save_keys)
    save_keys_button.grid(row=0, column=1, pady=5, padx=5)

    select_keys_button = ttk.Button(key_frame, text="Select
Keys", command=select_keys)
    select_keys_button.grid(row=0, column=2, pady=5, padx=5)

    # Encryption Frame
    encryption_frame = tk.LabelFrame(frame, text="Encryption",
padx=10, pady=10, borderwidth=0, highlightthickness=0)
    encryption_frame.grid(row=3, column=0, columnspan=2, pady=5,
sticky="ew")

    encrypt_button = ttk.Button(encryption_frame, text="Encrypt
File", command=encrypt_file)
    encrypt_button.grid(row=0, column=0, pady=5, padx=5)

    encrypt_text_button = ttk.Button(encryption_frame,
text="Encrypt Text", command=encrypt_text)
    encrypt_text_button.grid(row=0, column=1, pady=5, padx=5)

    # Decryption Frame
    decryption_frame = tk.LabelFrame(frame, text="Decryption",
padx=10, pady=10, borderwidth=0, highlightthickness=0)
    decryption_frame.grid(row=4, column=0, columnspan=2, pady=5,
sticky="ew")

    decrypt_button = ttk.Button(decryption_frame, text="Decrypt
File", command=decrypt_file)
    decrypt_button.grid(row=0, column=0, pady=5, padx=5)

    root.mainloop()
```