Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
в.о. завідувача кафедри
кібербезпеки та захисту
інформації
_____Іван ПАРХОМЕНКО
«____» червня 2023 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 «Кібербезпека»
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: Програмний засіб для забезпечення цілісності та конфіденційності даних за рахунок використання криптографічних механізмів

Виконавець: студентки 4 курсу, групи КБ-42

_____ **Руслана АЛЕКСЄЄВА**
(підпис) (ім'я прізвище)

	Прізвище, ініціали	**Підпис**
Керівник роботи	Андрій ФЕСЕНКО	

Нормоконтроль	Андрій БІГДАН	

Київ 2023

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри кібербезпеки
та захисту інформації
_____ Сергій ТОЛЮПА
«24» жовтня 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентці **КБ-42** **Руслана АЛЕКСЄЄВА**
(група) (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Програмний засіб для забезпечення цілісності та конфіденційності даних за рахунок використання криптографічних механізмів

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №3 від 20.10.2022 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

AES, алгоритми гешування, кваліфікований цифровий підпис, еліптична криптографія, програмування, ECDSA, Python, PyQt.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Аналіз необхідності безпечного обміну інформацією, порівняльний аналіз та дослідження криптографічних шифрування, аналіз підходів створення кваліфікованого електронного підпису, опис архітектури програмного забезпечення, криптографія на основі еліптичних кривих, програмний засіб.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність: є розроблений програмний засіб для забезпечення цілісності та конфіденційності даних за допомогою алгоритму шифрування AES та електронного цифрового підпису на основі еліптичних кривих.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 24 жовтня 2022 року

Завдання видав _____ (підпис) Андрій ФЕСЕНКО (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Руслана АЛЕКСЄЄВА (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	24.10.2022 – 23.11.2022	*виконано*
2	Аналіз літератури	23.11.2022 – 06.01.2023	*виконано*
3	Аналіз криптографічних механізмів	06.01.2023 – 20.01.2023	*виконано*
4	Дослідження криптографічних механізмів	20.01.2023 – 10.02.2023	*виконано*
5	Аналіз архітектури програмного застосунку	10.02.2023 – 25.02.2023	*виконано*
6	Програмна реалізація криптографічних механізмів	25.02.2023 – 18.03.2023	*виконано*
7	Розробка програмного засобу	18.03.2023 – 24.04.2023	*виконано*
8	Опис програмного засобу	24.04.2023 – 30.04.2023	*виконано*
9	Оформлення пояснювальної записки	30.04.2023 – 12.06.2023	*виконано*

Завдання видав _____ (підпис) Андрій ФЕСЕНКО (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Руслана АЛЕКСЄЄВА (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 12 червня 2023 року

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст займає 79 сторінок, включає в себе зміст, вступ, три розділи кваліфікаційної роботи, висновки та список джерел. Крім того, робота містить 1 додаток із загальною кількістю сторінок 3. У пояснювальній записці кваліфікаційної роботи міститься 70 рисунків, 6 таблиць та 29 літературних джерел.

Метою роботи є розробка програмного засобу для забезпечення цілісності та конфіденційності даних за допомогою блокового алгоритму шифрування та електронного цифрового підпису.

Об'єктом дослідження є процес забезпечення цілісності та конфіденційності даних в інформаційних системах.

Предметом дослідження є методи, для забезпечення цілісність та конфіденційність даних.

Методи дослідження:

- аналіз криптографічних механізмів;
- порівняння криптографічних алгоритмів;
- моделювання та обробка даних за допомогою програмного засобу написаного на мові програмування Python;

Практичною цінністю є розроблений програмний засіб для забезпечення цілісності та конфіденційності даних за допомогою алгоритму шифрування AES та електронного цифрового підпису на основі еліптичних кривих.

Тема: програмний засіб для забезпечення цілісності та конфіденційності даних за рахунок використання криптографічних механізмів.

Ключові слова: криптографія, крипостійкість, цілісність, конфіденційність, AES, ECDSA, ECC, Python.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕННЬ

КЕП	–	Кваліфікований Електронний Підпис
ANSI	–	American National Standards Institute
AES	–	Advanced Encryption Standard
CBC	–	Cipher Block Chaining
CFB	–	Cipher Feedback
DES	–	Data Encryption Standard
3-DES	–	Triple Data Encryption Standard
ECDSA	–	Elliptic Curve Digital Signature Algorithm
ECC	–	Elliptic Curve Cryptography
IDEA	–	International Data Encryption Algorithm
IEEE	–	Institute of Electrical and Electronics Engineers
NIST	–	National Institute of Standards and Technology
OFB	–	Output Feedback
RSA	–	Rivest-Shamir-Adleman
SEC	–	Standards for Efficient Cryptography
SP-мережа	–	мережа Substitution-Permutation (підстановки перестановки)
XOR	–	Exclusive OR

ВСТУП

За останні роки сфера цифрової інформації та обробки даних значно збільшилась і стала необхідною складовою сучасного суспільства. Однак, разом з розвитком цифрових технологій зростає й загроза безпеці цієї інформації. Тому актуальність дослідження, спрямованого на забезпечення цілісності та конфіденційності даних, надзвичайно висока. Забезпечення цілісності та конфіденційності даних є критичним завданням в різних галузях, таких як банківські системи, електронна комерція, медична сфера та багато інших. Існує необхідність у програмному засобі, який забезпечує надійний захист даних та зменшує ризик несанкціонованого доступу до них. Попередні технології шифрування та підписування даних не завжди здатні забезпечити необхідний рівень безпеки у сучасних умовах. Тому важливо розробити новий програмний засіб, який використовує блочний алгоритм шифрування та електронний цифровий підпис на основі еліптичних кривих. Цей новий підхід до захисту даних може забезпечити більшу ефективність і надійність, в той же час вимагаючи менше обчислювальних ресурсів. Застосування еліптичних кривих у блочному шифруванні та електронному цифровому підписі є інноваційним підходом, який знаходить все більше застосувань у сфері криптографії. Використання еліптичних кривих дозволяє забезпечити високий рівень безпеки при більш короткій довжині ключа порівняно з традиційними алгоритмами, що робить його особливо привабливим для використання в сучасних інформаційних системах.

Дана робота також вносить новизну в галузь криптографії, оскільки використовує поєднання блочного шифрування та електронного цифрового підпису на основі еліптичних кривих. Результати дослідження можуть сприяти подальшому розвитку методів захисту даних і створенню більш безпечних інформаційних систем. Тож *актуальність роботи* полягає в розробці рішення для безпечного обміну інформацією з використанням сучасних методів криптографії.

Метою роботи є розробка програмного засобу для забезпечення цілісності та конфіденційності даних за допомогою блокового алгоритму шифрування та електронного цифрового підпису.

Для досягнення зазначеної мети кваліфікаційної роботи поставлено наступні *завдання*:

- Розглянути основні принципи шифрування та електронного цифрового підпису.
- Дослідити теоретичні аспекти блочних алгоритмів шифрування та еліптичних кривих.
- Реалізувати вибраний блочний алгоритм шифрування та електронний цифровий підпис на основі еліптичних кривих у програмному засобі.
- Розробити програмний засіб для забезпечення цілісності та конфіденційності даних
- Провести тестування розробленого програмного засобу

Об'єктом дослідження є процес забезпечення цілісності та конфіденційності даних в інформаційних системах.

Предметом дослідження є методи, для забезпечення цілісність та конфіденційність даних.

Методи дослідження:

- аналіз відкритих джерел;
- дослідження наявних алгоритмів шифрування;
- моделювання, проектування та розробка програмного засобу;

Практичною цінністю є розроблений програмний засіб для забезпечення цілісності та конфіденційності даних за допомогою алгоритму шифрування AES та електронного цифрового підпису на основі еліптичних кривих.

Апробація роботи. Основні результати доповідалися та обговорювалися на:

Ruslana Aleksieieva, Andriy Fesenko, Andriy Dudnik, Yerlan Zhanerke. Software tool for ensuring data integrity and confidentiality through the use of cryptographic mechanisms. MoMLeT+DS 2023: 5th International Workshop on Modern Machine Learning Technologies and Data Science, 3 червня 2023, Львів. Матеріали прийняті до

друку в журнал, що індексується в науково-метричних базах: Scopus, DBLP, Google Scholar.

Алексєєва Р., Фесенко А. Аналіз доцільності використання ECDSA. Проблеми кібербезпеки інформаційно-телекомунікаційних систем (PCSITS) : VI МІЖНАР. НАУКОВО-ПРАКТ. КОНФ., м. Київ, 27 квіт. 2023 р. Київ, 2023. С. 2–3.

РОЗДІЛ 1
АНАЛІЗ ТА ДОСЛІДЖЕННЯ КРИПТОГРАФІЧНИХ МЕХАНІЗМІВ

1.1 Аналіз необхідності обміну інформацією за допомогою програмного забезпечення

У наші дні люди значною мірою покладаються на технології та цифрові комунікації. Це призвело до зростаючої потреби в безпечному обміну інформацією. В даний час тільки криптографія дає можливість забезпечити конфіденційність, цілісність і достовірність даних під час передачі. Криптографія дозволяє користувачам безпечно спілкуватися, унеможливлюючи несанкціонований доступ.

1.1.1 Необхідність безпечного обміну інформацією

Захист даних необхідний з кількох причин.

1. Конфіденційна інформація, така як особисті дані, повинна бути захищена від несанкціонованого доступу, перехоплення та модифікації даних.

2. Компанії повинні захищати свою інтелектуальну власність, комерційну таємницю та конфіденційні документи, які циркулюють у компанії.

3. У свою чергу державні органи зобов'язані захищати інформацію з грифами таємності

4. Безпечний обмін інформацією має вирішальне значення для підтримки довіри та впевненості в безпеці цифрових комунікацій

1.1.2 Роль криптографії в безпечному обміні інформацією

Криптографія відіграє важливу роль у забезпеченні безпечного обміну інформацією. Завдяки шифруванню даних за допомогою криптографічних алгоритмів інформація захищена від несанкціонованого доступу, перехоплення та модифікації.

Використовуючи криптографію, ви також можете забезпечити цілісність і автентичність даних за допомогою цифрових підписів і геш-функцій. Крім того, криптографія може використовуватися для забезпечення безпечного обміну ключами та сертифікатами, які необхідні для безпечного зв’язку.

Для цього використовується відповідне програмне забезпечення, яке сприяє підвищенню рівня безпеки даних. Ці програми включають криптографічні засоби захисту інформації та можуть містити такі функції, як створення сертифікатів або ключів, шифрування та дешифрування повідомлень або файлів, створення та перевірка кваліфікованих цифрових підписів для підтримки цілісності та автентичності документів.

Підсумовуючи, безпечний обмін інформацією є критично важливим у сучасну цифрову епоху. Криптографія є потужним інструментом для забезпечення конфіденційності, цілісності та автентичності даних, що передаються. Використовуючи криптографічні алгоритми, компанії, організації, установи та звичайні люди можуть захистити свою конфіденційну інформацію. За словами Брюса Шнайера, криптографія не є ідеальним рішенням, але вона все ще є найкращим доступним інструментом для забезпечення безпечного обміну інформацією в час цифрових технологій.

1.2 Порівняльний аналіз блокових алгоритмів шифрування

Зараз існує багато різних алгоритмів шифрування, зокрема блокові алгоритми шифрування широко використовуються для захисту інформації. Розглянемо порівняльну характеристику найпопулярніших блокових симетричних алгоритмів. На рисунку 1.1 зображені п’ять алгоритмів, які будуть аналізуватися. Для порівняння були обрані такі симетричні блокові алгоритми як DES (Стандарт шифрування даних), 3-DES (Потрійний стандарт шифрування даних), AES (Розширений стандарт шифрування), IDEA (Міжнародний алгоритм шифрування даних), BlowFish.

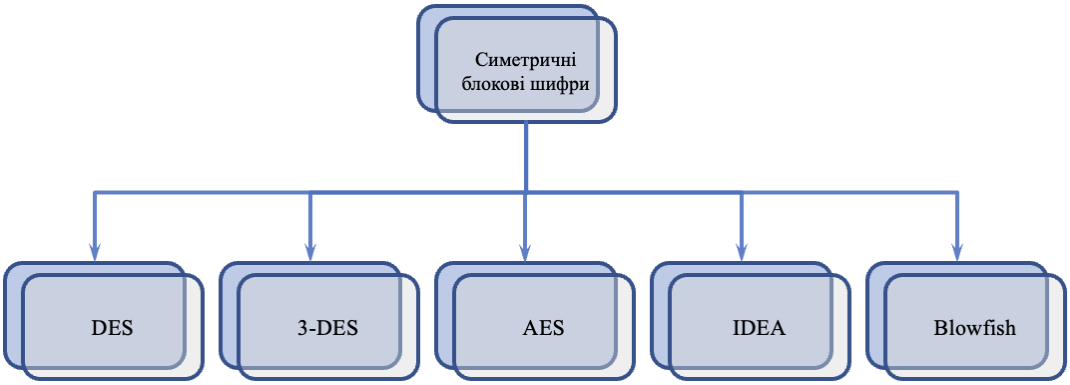

Рисунок 1.1 – Симетричні блокові шифри

1.2.1 Критерії порівняння

Аналіз алгоритмів можна проводити за багатьма критеріями, розглянемо основні з них, які зображені на рисунку 1.2.

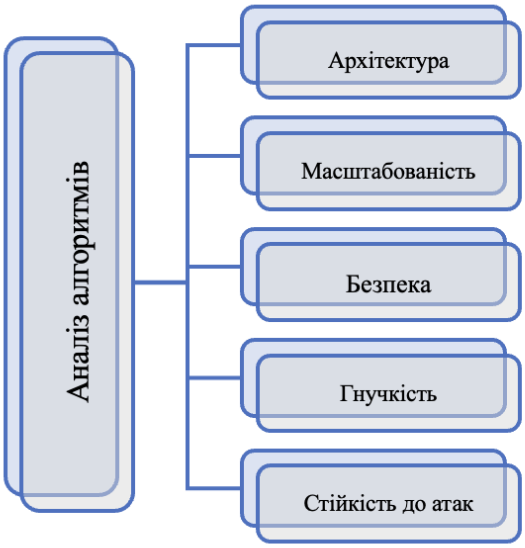

Рисунок 1.2 – Критерії порівняння алгоритмів

1. Архітектура. Архітектура включає в себе основні характеристики алгоритму, розмір ключа, тип мережі, функції, операції, які використовуються, кількість раундів тощо. Отже, архітектура складає базову структуру алгоритму [1].

2. Масштабованість. Масштабованість використовується для аналізу продуктивності алгоритму на основні таких параметрів як споживання пам’яті,

швидкість шифрування, продуктивність програмно-апаратного забезпечення; Обчислювальна ефективність, режим, тип файлу даних, швидкість, пропускна здатність тощо.

3. Безпека. Безпека основний критерій оцінки алгоритму. Він визначає наскільки добре алгоритм протистоїть атакам. Для цього будемо використовувати наступні характеристики: довжина ключа (чим більша довжина ключа, тим складніше його зламати), стійкість до криптоаналізу тощо.

4. Гнучкість. Гнучкість визначає, чи може алгоритм витримувати незначні модифікації відповідно до потреб та вимог.

5. Стійкість до атак. Стійкість до атак визначає, наскільки ефективно працює алгоритм, використовуючи доступні йому комп'ютерні ресурси, та наскільки він вразливий до різних типів атак.

1.2.2 Опис алгоритмів для порівняння

Стандарт шифрування даних (DES)

Стандарт шифрування даних (DES) був спільно розроблений у 1974 році IBM та урядом США, щоб встановити стандарт, який кожен міг використовувати для безпечного обміну даними один з одним. Він працює з блоками з 64 бітів, використовуючи секретний ключ довжиною 56 бітів. DES базується на шифрі, відомому як блоковий шифр Фейстеля. Він складається з 16 раундів, де кожен раунд містить перетасування бітів, нелінійні підстановки (S-блоки) та операції виняткового АБО [2].

Потрійний стандарт шифрування даних (3DES)

У криптографії Triple DES (3DES) - це загальна назва блокового шифру з алгоритмом потрійного шифрування даних (TDEA або Triple DEA), який тричі застосовує алгоритм шифрування стандарту шифрування даних (DES) до кожного блоку даних. Він використовує три 56-бітні ключі DES, що забезпечує загальну довжину ключа 168 біт [2].

Розширений стандарт шифрування (AES)

AES (або Rijndael) з’явився як потужна заміна DES під час конкурсу, проведеного Національним інститутом стандартів і технологій (NIST). Rijndael: алгоритм, розроблений Daemen і Rijmen, було визнано найкращим і оголошено новим AES. NIST обирає Rijndael завдяки його простоті та високій продуктивності. Він швидкий, компактний і має дуже просту математичну структуру.

AES – це симетричний блоковий шифр із розміром блоку 128 біт. Довжина ключа може бути 128 біт, 192 біт або 256 біт; називається AES-128, AES-192 і AES-256 відповідно. AES-128 використовує 10 раундів, AES-192 використовує 12 раундів, а AES-256 використовує 14 раундів [3].

Міжнародний алгоритм шифрування даних (IDEA)

IDEA – Міжнародний алгоритм шифрування даних — це симетричний блоковий шифр, розроблений Сюецзя Лаєм і Джеймсом Мессі зі Швейцарського федерального технологічного інституту в 1990 році. IDEA — це один із ряду звичайних алгоритмів шифрування, запропонованих останніми роками для заміни DES [4].

Blowfish

Blowfish — це симетричний блочний шифр, розроблений у 1993 році Брюсом Шнайером і включений до великої кількості набори шифрів і продукти шифрування. Blowfish — це ключ змінної довжини, 64-бітний блоковий шифр [5].

1.2.3 Порівняння блокових алгоритмів

Архітектура

Характеристики за архітектурою алгоритмів – це базові характеристики алгоритму, наглядно зображені у таблиці 1.1 [1].

Таблиця 1.1

Порівняння за архітектурою

Характеристики	DES	3DES	IDEA	AES	Blowfish
Рік створення	1974	1998	1991	2000	1993

Продовження табл. 1.1

Довжина ключа	56	112, 168	128	128, 192, 256	32 до 448
Тип шифру	Симетричні блокові алгоритми				
Використані функції	IP, IP-1, E, P, S-Box, PC-1, PC-2	IP, IP-1, E, P, S-Box, PC-1, PC-2	-	S-Box, GF(2^8)	S-Box
Використані операції	XOR, <<<, >>>	XOR, <<<, >>>	XOR, додавання 2^{16}, множення 2^{16+1}	XOR, <<<, >>>	+, XOR, <<<, >>>
Структура алгоритму	Мережа Фейстеля	Мережа Фейстеля	Схема Лая-Мессі	Мережа підстановки-перестановки (SP-мережа)	Мережа Фейстеля

Масштабованість/продуктивність

Система є масштабованою, якщо її продуктивність залишається стабільною навіть за критичного сценарію. Продуктивність будь-якої системи можна оцінити за певними критеріями. Такі критерії, фактори або параметри відомі як показники ефективності. Порівняння за масштабованістю наведено у таблиці 1.2.

Таблиця 1.2

Порівняння за масштабованістю

Характеристики	DES	3DES	IDEA	AES	Blowfish
Час на зашифрування	Високий	Високий	Низький	Високий	Високий
Час на розшифрування	Середній	Середній	Високий	Високий	Низький
Швидкість обчислень	Швидко	Помірний	Швидко	Швидко	Дуже швидко
Пропускна здатність	Середня	Низька	Висока	Висока	Дуже висока

Безпека

Криптографічна безпека визначає, чи безпечна схема шифрування проти грубої сили та різних атак із простим шифром. Шифр використовується для забезпечення захисту від небажаного розголошення відкритого тексту. Будь-який криптоаналітик або хакер зламує шифр з метою відновлення відкритого тексту. Якщо хакер відновить секретний ключ, він зможе прочитати всі повідомлення після цього так само швидко, як і законний користувач. Про безпеку завжди говорять відносно загроз. Якщо припустити, що зловмисники мають доступ до всього через незахищений канал – це Інтернет, безпеку шифру можна оцінити після розгляду обчислювальних можливостей хакера [1].

Припущення Керкгофа: криптоаналітик знає повний процес шифрування та дешифрування, за винятком значення секретного ключа. Це означає, що безпека системи шифрування секретного ключа повністю залежить від секретного ключа.

Порівняння за безпекою наведено у таблиці 1.3.

Таблиця 1.3

Порівняння за безпекою

Характеристики	DES	3DES	IDEA	AES	Blowfish
Пари відкритий / зашифрований текст (диференціальний криптоаналіз 2^розмір блоку)	2^64	2^64	2^128	2^64	2^64
Складність обробки (2^ розмір ключа)	7.2 x 10^16 (2^56)	5.1 x 10^33 (2^112) to 3.74 x 10^50 (2^168)	3.4 x 10^38 (2^ 128) to 1.15 x 10^77(2^256)	3.4 x 10^38 (2^128)	4.29 x 10^9 (2^32) to 7.26 x 10^134 (2^448)
Безпека	Вразливий	Вразливий	Висока	Висока	Висока

Продовження табл. 1.3

Стійкість до криптоаналізу	Вразливий до лінійного та диференціального криптоаналізу, грубої сили	Вразливий до різної грубої сили. Зловмисники можуть аналізувати plain text, вразливі до обраного plain text, Відомий plain text	Вразливість до слабких ключів	Стійкий до різних атак, лінійної інтерполяції та квадратних атак, вразливий до обраного plain text	Вразливий до різної атаки грубої сили Атака за словником
Стійкість до статистичних атак	Середня	Висока	Висока	Висока	Висока

Гнучкість

Гнучкість алгоритму шифрування можна визначити як здатність алгоритму шифрування пристосовуватися до різноманітних потреб користувачів і контексту використання. Гнучкість алгоритму шифрування може включати в себе можливість використання різних типів ключів, різних режимів роботи, підтримку різних довжин ключів, можливість налаштування параметрів шифрування та інші функції [1].

Порівняння за гнучкістю наведено у таблиці 1.4.

Стійкість до атак

Стійкість до атак - це властивість алгоритмів шифрування, яка характеризує їх здатність захистити вхідні дані від несанкціонованого доступу або зламу.

Порівняння за стійкістю до атак наведено у таблиці 1.5.

Таблиця 1.4

Порівняння за гнучкістю

Характеристики	DES	3DES	IDEA	AES	Blowfish
Гнучкість	-	+	-	+	+
Пояснення	Жодна зміна не дозволяється	3 рази повторення	Жодна зміна не дозволяється	Структура гнучка до числа, кратного 64.	Довжина ключа має бути кратною 32 бітам

Таблиця 1.5

Порівняння за стійкістю до атак

DES	3DES	IDEA	AES	Blowfish
Дуже вразливий до лінійного криптоаналізу, слабких ключів і грубої сили. Для зламу ключа потрібно лише 2^56 комбінацій. Він розроблений для апаратного забезпечення, тому працює повільно для програмного забезпечення.	Піддається диференціальним і пов’язаним ключовим атакам. Також сприйнятливий до певних варіацій зустрічної атаки посередині.	Сприйнятливі до різних класів слабких ключів і дуже вразливі до атак на ключі, як-от атаки на розклад ключа та пов’язані атаки на диференціальний час ключа. Перші три раунди алгоритму IDEA спостерігається для атак із диференціальним часом пов’язаного ключа та атак із розкладом ключа.	Відсутність серйозної слабкості; Деякі початкові раунди AES спостерігаютьс я незахищеними, тобто початковий раунд може зламати квадратним методом.	Вразливі до слабких ключів, 4 раунди піддаютьс я диференц іальним атакам 2-го порядку.

Проведено порівняльний аналіз кількох блокових шифрів за різними параметрами, такими як безпека, гнучкість, стійкість до атак тощо. Архітектура та робота всіх алгоритмів були розглянуті. Кожен алгоритм має свої переваги відповідно до різних критеріїв. Видно, що міцність кожного алгоритму шифрування залежить від керування ключами, типу криптографії, кількості ключів, кількості бітів, що використовуються в ключі. Чим довша довжина ключа та довжина даних, тим більше буде споживана потужність. Тому не рекомендується використовувати коротку послідовність даних і довжину ключа. Усі ключі засновані на математичних властивостях, і їх сила зменшується з часом. Ключі з більшою кількістю бітів вимагають більше часу обчислення, що просто вказує на те, що системі потрібно більше часу для шифрування даних.

Можемо зробити висновок, що алгоритм Blowfish є кращим щодо пропускної здатності, енергоспоживання та часу обробки порівняно з іншими алгоритмами. Але алгоритм AES зі змінною довжиною ключа та різною кількістю раундів є ефективним вибором для безпеки та гнучкості. AES є універсальним рішенням та з урахуванням всіх критеріїв переваги компенсують недоліки. Тому у подальшому досліджені буде використовуватися саме AES.

1.3 Алгоритм AES

1.3.1 Специфікація алгоритму AES

Для алгоритму AES довжина вхідного блоку, вихідного блоку та стану становить 128 біт. Для алгоритму AES довжина ключа шифру K становить 128, 192 або 256 біт. Довжина ключа представлена Nk = 4, 6 або 8, що відображає кількість 32-розрядних блоків (кількість стовпців) у ключі шифру [6].

Для алгоритму AES кількість раундів, які необхідно виконати під час виконання алгоритму, залежить від розміру ключа. Кількість раундів позначається Nr, де Nr = 10, якщо Nk = 4; Nr = 12, коли Nk = 6; і Nr = 14, коли Nk = 8. Принцип роботи алгоритму AES, зображено на рисунку 1.3.

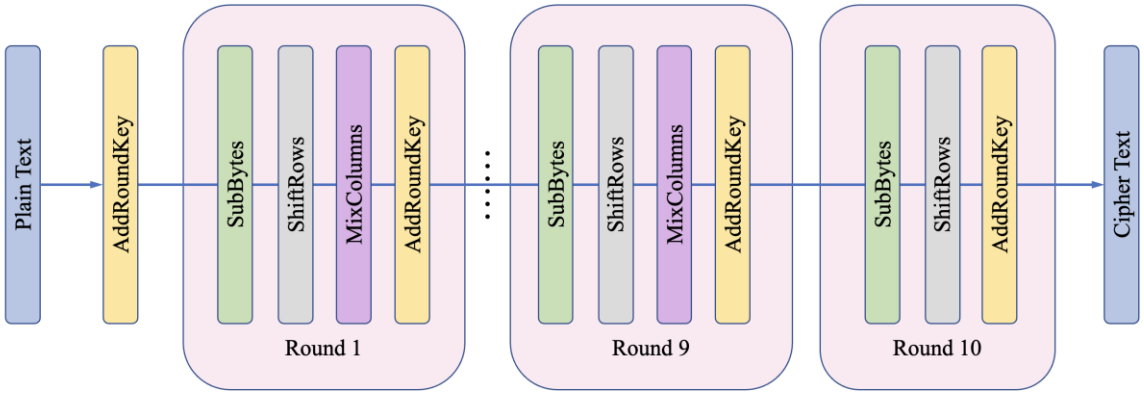

Рисунок 1.3 – Принцип роботи алгоритму AES

Алгоритм AES використовує раундову функцію, яка складається з чотирьох різних байт-орієнтованих перетворень, що зображені на рисунку 1.4:

1) SubBytes: нелінійна заміна байтів за допомогою таблиці підстановок (S-box)
2) ShiftRows: циклічне зміщення рядків масиву State на різну кількість байтів
3) MixColumns: змішування даних у кожному стовпці масиву стану
4) AddRoundKey: додавання раундового ключа до стану за допомогою простої побітової операції XOR [7].

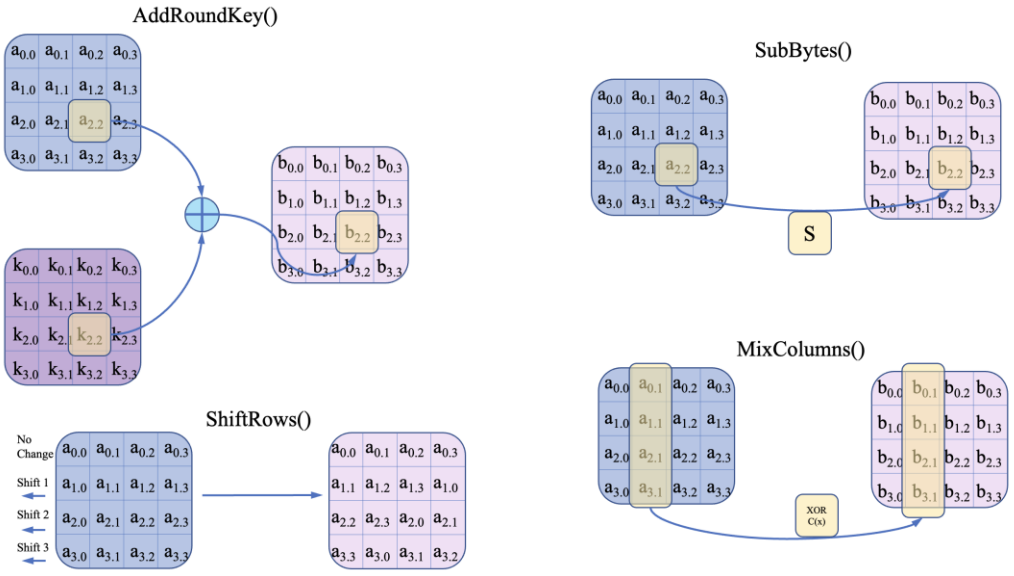

Рисунок 1.4 – Функції перетворення AES

Після початкового додавання ключа раунду масив перетворюється шляхом реалізації функції раунду 10, 12 або 14 разів (залежно від довжини ключа), причому остаточний раунд дещо відрізняється від перших Nr - 1 раундів. Усі раунди ідентичні, за винятком останнього раунду, який не включає перетворення MixColumns(). Остаточний стан потім копіюється на вихід.

1.3.2 Режими роботи блокових шифрів

Режим зчеплення блоків зашифрованих даних (CBC). У цьому режимі виконання вхідними даними для криптографічного алгоритму є XOR наступного блоку незашифрованих даних із попереднім блоком зашифрованих даних [8, 9].

Режим зворотного зв'язку шифру (CFB). У цьому режимі виконання кожен виклик алгоритму обробляє l-бітні вхідні значення. Попередній зашифрований блок використовується як вхідні дані для алгоритму; операція XOR застосовується до 1-бітового виходу алгоритму та наступного незашифрованого 1-бітового блоку, результатом якого є наступний 1-бітовий зашифрований блок [8, 9].

Режим вихідного зворотного зв'язку (OFB). Цей метод виконання подібний до методу виконання CFB, за винятком того, що результат шифрування попереднього блоку використовується як вхід алгоритму при шифруванні наступного блоку; лише після цього операція XOR виконується над наступним l-бітом незашифровані дані [8, 9].

1.4 Еліптична криптографія

Еліптична криптографія (ECC) - це підхід до криптографії з відкритим ключем, заснований на алгебраїчній структурі еліптичних кривих над кінцевими полями. ECC дозволяє використовувати менші ключі порівняно з не еліптичною криптографією (на основі простих полів Галуа), щоб забезпечити еквівалентну безпеку [10, 11]. Приклад еліптичної кривої зображений на рисунку 1.5.

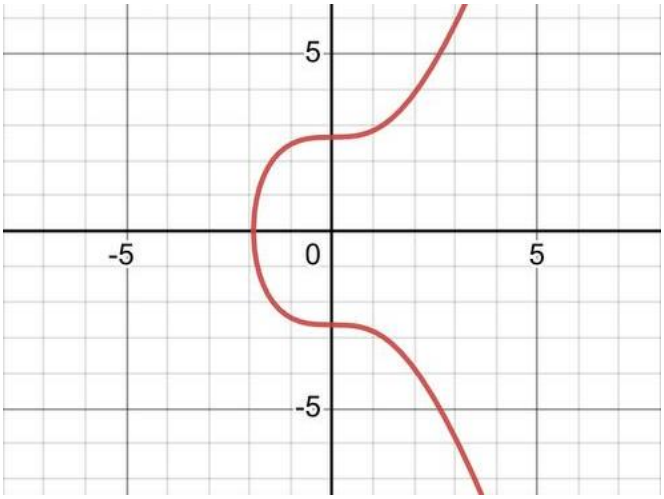

Рисунок 1.5 – Графік еліптичної кривої

Еліптичні криві застосовні для узгодження ключів, цифрових підписів, псевдовипадкових генераторів та інших завдань. Непрямо їх можна використовувати для шифрування шляхом поєднання узгодженості ключів із симетричною схемою шифрування.

Криптографія з відкритим ключем заснована на складності певних математичних проблем. Ранні системи з відкритим ключем базували свою безпеку на припущенні, що важко розкласти велике ціле число, що складається з двох або більше великих простих множників. Для більш пізніх протоколів на основі еліптичної кривої базовим припущенням є те, що знайти дискретний логарифм випадкового елемента еліптичної кривої щодо загальновідомої базової точки неможливо: це «проблема дискретного логарифму еліптичної кривої» (ECDLP). Безпека криптографії еліптичної кривої залежить від здатності обчислити множення на точку та неможливості обчислити множене, враховуючи початкові точки та точки добутку. Розмір еліптичної кривої, виміряний загальною кількістю дискретних цілих пар, що задовольняють рівнянню кривої, визначає складність проблеми [11].

Основна перевага, яку обіцяє криптографія на основі еліптичних кривих, - це менший розмір ключа, що зменшує вимоги до зберігання та передачі, тобто група еліптичної кривої може забезпечувати той самий рівень безпеки, який забезпечує система на основі RSA з великим модулем і, відповідно, більшим ключем: наприклад, 256-бітний відкритий ключ еліптичної кривої повинен забезпечувати порівнянну

безпеку з 3072-бітним відкритим ключем RSA. Порівняння криптостійкості даних алгоритмів було розглянуто у розділі 1.5.3 [10].

1.4.1 Операції над точками еліптичної кривої

Скалярне множення еліптичної кривої — це операція послідовного додавання точки вздовж еліптичної кривої до самої себе. Він використовується в криптографії з еліптичною кривою (ECC) як засіб створення односторонньої функції. Широко поширеною назвою для цієї операції також є множення точок еліптичної кривої, але це може створити неправильне враження множення двох точок [12] (рисунок 1.6.).

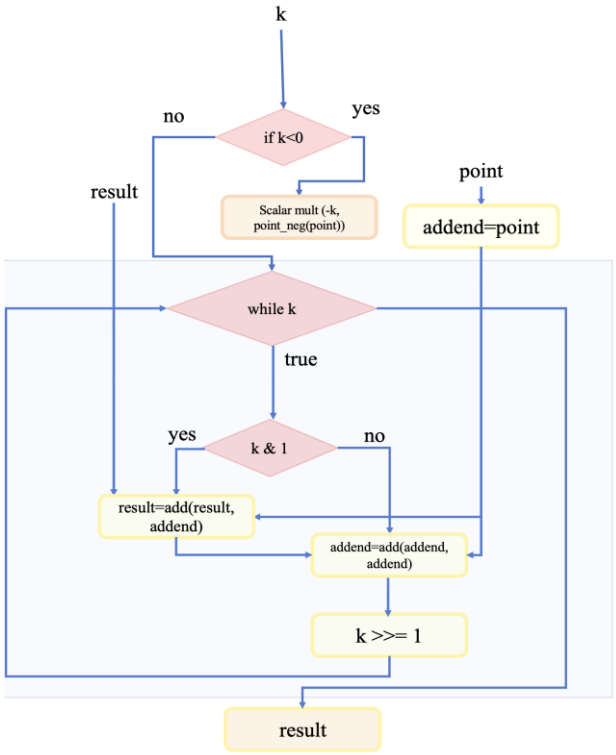

Рисунок 1.6 – Скалярне множення точок еліптичної кривої

Отже, дано криву E, визначену вздовж деякого рівняння в скінченному полі (наприклад, $E: y^2 = x^3 + ax + b$), множення точки визначається як повторне додавання точки вздовж цієї кривої. Позначимо як $nP = P + P + P + \cdots + P$ для деякого скаляра (цілого) n і точки $P = (x, y)$ яка лежить на кривій E. Цей тип кривої відомий як крива Вейєрштрасса.

Безпека сучасного ECC залежить від складності визначення n з $Q = nP$ за відомих значень Q і P, якщо n велике (відома як проблема дискретного логарифмування еліптичної кривої за аналогією з іншими криптографічними системами). Це пояснюється тим, що додавання двох точок на еліптичній кривій (або додавання однієї точки до самої себе) дає третю точку на еліптичній кривій, розташування якої не має безпосереднього очевидного зв'язку з розташуванням перших двох, і повторення цього багато разів дає точку nP, яка може бути по суті будь-де. Повернути цей процес назад, тобто враховуючи $Q = nP$ і P, і визначити n можна лише спробувавши всі можливі n — зусилля, яке обчислювально нерозв'язне, якщо n велике [12].

Існують три загальноприйняті операції для точок еліптичної кривої: додавання, подвоєння та заперечення [13, 14].

1. Точка заперечення

Заперечення точки — це знаходження такої точки, що додавання її до себе призведе до нескінченності (рисунок 1.7).

$$P + (-P) = \infty \tag{1.1}$$

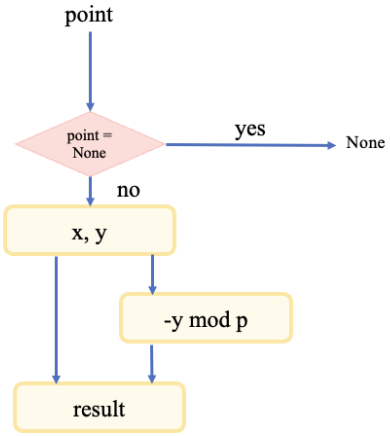

Рисунок 1.7 – Заперечення точки

Для еліптичних кривих форми $E: y^2 \equiv x^3 + ax + b$ запереченням є точка з тією самою координатою x, але запереченою координатою y:

$$(x, y) + \big(-(x, y)\big) = \infty \tag{1.2}$$

$$(x, y) + (x, -y) = \infty \quad (1.3)$$

$$(x, y) = -(x, y) \quad (1.4)$$

2. Додавання точок

З 2 різними точками, P і Q, додавання визначається як заперечення точки, що виникає в результаті перетину кривої, E, і прямої, визначеної точками P і Q, що дає точку, R [14].

$$P + Q = R \quad (1.5)$$

$$(x_p, y_p) + (x_q, y_q) = (x_r, y_r) \quad (1.6)$$

Якщо припустити, що еліптична крива E задана як $E: y^2 \equiv x^3 + ax + b$, це можна обчислити як:

$$\lambda = \frac{y_q - y_p}{x_q - x_p} \quad (1.7)$$

$$x_r = \lambda^2 - x_p - x_q \quad (1.8)$$

$$y_r = \lambda(x_p - x_r) - y_p \quad (1.9)$$

Число λ є кутовим коефіцієнтом січної, проведеної через точки P та Q.

Ці рівняння правильні, якщо жодна з точок не є нескінченною точкою, і якщо точки мають різні координати x (вони не є взаємними зворотними). Це важливо для алгоритму перевірки ECDSA, де геш-значення може бути нульовим.

3. Подвоєння точки

Якщо точки P і Q збігаються (з однаковими координатами), додавання відбувається аналогічно, за винятком того, що немає чітко визначеної прямої лінії, що проходить через P, тому операція замикається з використанням граничного випадку, дотичної до кривої E, на P [13].

Це розраховується, як зазначено вище, використовуючи похідні $(dE/dx)/(dE/dy)$:

$$\lambda = \frac{3x_p^2 + a}{2y_p} \quad (1.10)$$

де a – із визначального рівняння кривої E, наведеної вище.

Алгоритм додавання та подвоєння точок еліптичної кривої зображений на рисунку 1.8.

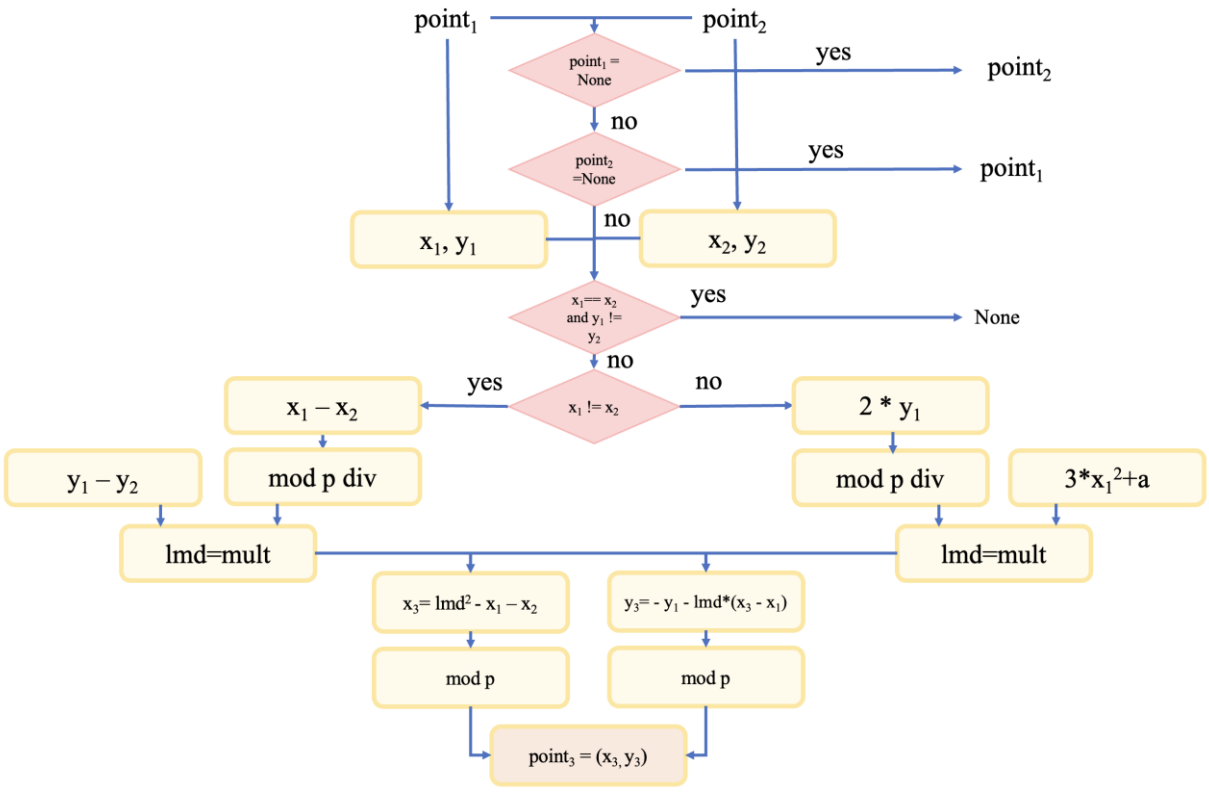

Рисунок 1.8 – Додавання та подвоєння точок еліптичної кривої

Графіки подвоєння та додавання точок еліптичної кривої зображенні відповідно на рисунках 1.9 та 1.10.

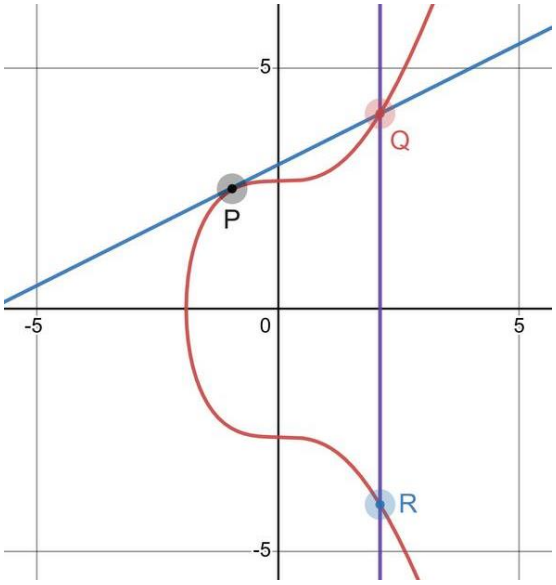

Рисунок 1.9 – Подвоєння точок еліптичної кривої P+P=R

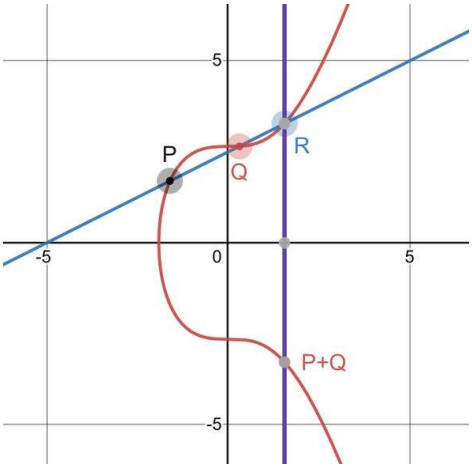

Рисунок 1.10 – Додавання точок еліптичної кривої

1.4.2 Створення ключів на основі еліптичних кривих

Першочергово обирається крива з відповідними параметрами:

$$T = (p, a, b, G, n, h), \tag{1.10}$$

що складається з цілого числа p, що задає скінченне поле F_p, двох елементів $a, b \in F_p$, що задають еліптичну крива $E(F_p)$, що визначається рівнянням: $E: y^2 \equiv x^3 + ax + b(modp)$, базова точка $G = (x_G, y_G)$ на $E(F_p)$, просте n, яке є порядком G, і ціле число h, яке є співмножником $h = E(F_p)/n$ [14].

Параметри області еліптичної кривої над F_p точно визначають еліптичну криву та базову точку. Це необхідно для точного визначення криптографічних схем із відкритим ключем на основі ECC.

Відповідно на виході з алгоритму ми маємо отримати пару ключів, що складається з закритого ключа d (випадково обраного ціле число з інтервалу $[1, n-1]$) і відкритого ключа еліптичної кривої $Q = (xQ, yQ)$, який є точкою $Q = dG$.

Пари ключів еліптичної кривої мають бути згенеровані таким чином [14]:

- Вхідні дані: дійсні параметри області еліптичної кривої $T = (p, a, b, G, n, h)$.
- Вихід: пара ключів еліптичної кривої (d, Q), пов’язана з кривою
- Дії: Створення пари ключів еліптичної кривої наступним чином:

1. Випадково або псевдовипадково вибрати ціле число d з інтервалу $[1, n-1]$.

2. Обчислення $Q = dG$.

3. Результат: (d, Q)

Алгоритм створення пари ключів зображений на рисунку 1.11.

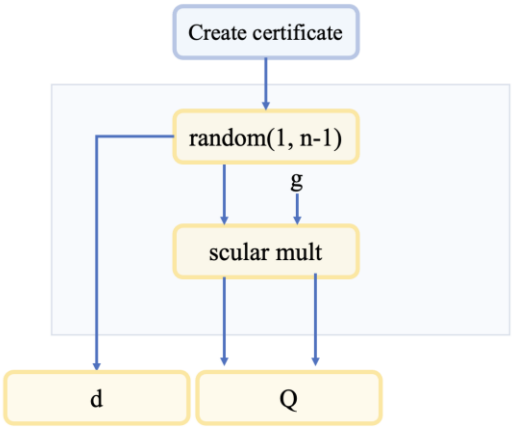

Рисунок 1.11 – Створення ключів

1.5. Аналіз підходів створення КЕП

Сьогодні використання криптографічних засобів продовжує набирати обертів. Криптографічні засоби захисту інформації, як кваліфіковані електроні підписи, часто використовуються для підпису важливих електронних документів, оскільки необхідна обережність для захисту конфіденційності чутливих даних, яка є незмінним атрибутом кожного документа. Тому важливою проблемою електронного підпису є вдосконалення алгоритму підпису та методу відновлення конфіденційних даних із самого підпису. Алгоритми еліптичних кривих також слід розглянути для вдосконалення для побудови більш ефективних асиметричних криптографічних алгоритмів [15].

1.5.1 Нормативно-правова база кваліфікованого електронного підпису

Згідно закону України про електроні довірчі послуги наведені основні терміни та щодо кваліфікованого електронного підпису [16].

Кваліфікований електронний підпис - удосконалений електронний підпис, який створюється з використанням засобу кваліфікованого електронного підпису і базується на кваліфікованому сертифікаті відкритого ключа;

Автентифікація - електронна процедура, яка дає змогу підтвердити електронну ідентифікацію фізичної, юридичної особи, інформаційної або інформаційно-комунікаційної системи та/або походження та цілісність електронних даних;

Ідентифікація особи - процедура використання ідентифікаційних даних особи з документів, створених на матеріальних носіях, та/або електронних даних, в результаті виконання якої забезпечується однозначне встановлення фізичної, юридичної особи або представника юридичної особи;

Електронні дані - будь-яка інформація в електронній формі;

Суб’єктами відносин у сфері електронних довірчих послуг є:

- користувачі електронних довірчих послуг;
- надавачі електронних довірчих послуг;
- органи з оцінки відповідності;
- засвідчувальний центр;
- центральний засвідчувальний орган;
- контролюючий орган.

1.5.2 Алгоритми для створення КЕП

Для подальшого порівняння представлені два алгоритми для створення КЕП, такі як ECDSA та RSA.

ECDSA – (Elliptic Curve Digital Signature Algorithm) – алгоритм з відкритим ключем для створення цифрового підпису, аналогічний за своєю будовою DSA, але визначений, на відміну від нього, не над кільцем цілих чисел, а в групі точок еліптичної кривої [17].

ECDSA (Elliptic Curve Digital Signature Algorithm) – перший алгоритм шифрування з відкритим ключем на базі точок еліптичної кривої в скінченному полу Галуа.

Цей алгоритм був прийнятий стандартом ISO в 1998 році, в 1999 стандартом ANSI, також в 2000 році — в IEEE та NIST.

Криптосистема RSA - це криптографічний алгоритм з відкритим ключем, заснований на складному завданні обчислення розкладання на множники, або пошуку множників довгих чисел. Алгоритм шифрування названий на честь трьох винахідників – Рона Ріввеста, Аді Шаміра та Леонарда Адлемана. Алгоритм шифрування RSA став першим методом, придатним для шифрування та КЕП. RSA — це система з відкритим або відкритим ключем. Алгоритм був розроблений в 1977 році [18, 19].

1.5.3 Порівняння криптостійкості алгоритмів на основі еліптичних кривих відносно на базі цілих чисел

ECC вимагає меншої довжини ключа, щоб забезпечити такий же рівень безпеки, як і довгі ключі RSA. Наведено дані в таблиці 1.6 про довжину ключа, необхідну для різних алгоритмів, для забезпечення необхідного рівня стійкості [10].

Таблиця 1.6

Порівняння довжини ключа відносно криптостійкості

Криптостійкість (в бітах)	RSA довжина публічного ключа (в бітах)	ECDSA довжина публічного ключа (в бітах)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Порівняно з RSA ECDSA більш безпечний проти сучасних методів злому завдяки своїй складності. ECDSA забезпечує такий же рівень безпеки, як і RSA, але

використовує меншу довжину ключа. Таким чином, довші ключі ECDSA займають більше часу, щоб зламати за допомогою bruteforce [20].

Ще одна перевага ECDSA над RSA є в продуктивності та масштабованості. Оскільки ECC забезпечує оптимальну безпеку з меншою довжиною ключа, він потребує менше обчислювальної потужності. Це корисно для пристроїв з обмеженою пам’яттю та обчислювальною потужністю.

1.6 КЕП

Цифровий підпис — це математичний метод, який використовується для перевірки автентичності та цілісності цифрового документа, повідомлення або програмного забезпечення. Це цифровий еквівалент власноручного підпису чи печатки, але він забезпечує набагато більшу безпеку. Цифровий підпис призначений для вирішення проблеми фальсифікації та видавання себе за іншу особу в цифрових комунікаціях [21].

1.6.1 Створення цифрового підпису

Цифрові підписи можуть надавати докази походження, ідентичності та статусу електронних документів, транзакцій або цифрових повідомлень. Підписувачі також можуть використовувати їх для підтвердження інформованої згоди. У багатьох країнах, включаючи США, цифрові підписи вважаються юридично зобов’язуючими так само, як традиційні рукописні підписи документів.

Цифрові підписи засновані на криптографії з відкритим ключем, також відомої як асиметрична криптографія. За допомогою алгоритму відкритого ключа генеруються два ключі, створюючи математично пов’язану пару ключів: один приватний і один публічний.

Технологія цифрового підпису вимагає від усіх сторін впевненості в тому, що особа, яка створює зображення підпису, зберегла секретний ключ. Якщо хтось інший

має доступ до приватного ключа підпису, ця сторона може створити шахрайські цифрові підписи від імені власника приватного ключа.

Щоб створити цифровий підпис, програмне забезпечення для підпису, наприклад програма електронної пошти, використовується для надання одностороннього гешу електронних даних, які потрібно підписати.

Геш — це рядок із буквами та цифрами фіксованої довжини, згенерований алгоритмом. Для шифрування гешу використовується особистий ключ творця цифрового підпису. Зашифрований геш разом з іншою інформацією, як-от алгоритм гешування, є цифровим підписом [21].

Причина шифрування гешу замість усього повідомлення чи документа полягає в тому, що геш-функція може перетворити довільний вхід у значення фіксованої довжини, яке зазвичай набагато коротше. Це економить час, оскільки гешування відбувається набагато швидше, ніж підписання.

Значення гешу є унікальним для гешованих даних. Будь-яка зміна в даних, навіть модифікація одного символу, призводить до іншого значення. Цей атрибут дозволяє іншим використовувати відкритий ключ підписувача для розшифровки гешу для перевірки цілісності даних.

Якщо розшифрований геш відповідає другому обчисленому гешу тих самих даних, це доводить, що дані не змінилися з моменту їх підписання. Але якщо два геші не збігаються, це означає, що дані були підроблені якимось чином і скомпрометовані, або підпис було створено за допомогою закритого ключа, який не відповідає відкритому ключу, наданому підписувачем. Це сигналізує про проблему з автентифікацією.

Алгоритм створення та перевірки цифрового підпису зображений на рисунку 1.12.

Цифровий підпис можна використовувати з будь-яким видом повідомлення, незалежно від того, зашифроване воно чи ні, просто щоб одержувач міг бути впевнений в ідентичності відправника та в тому, що повідомлення надійшло неушкодженим. Цифрові підписи ускладнюють для підписувача заперечення того, що

він щось підписав, оскільки цифровий підпис є унікальним як для документа, так і для підписувача, і він пов’язує їх разом. Ця властивість називається неспростовністю.

Цифрові підписи широко використовуються для підтвердження автентичності, цілісності даних і неспростування повідомлень і транзакцій, що здійснюються через Інтернет.

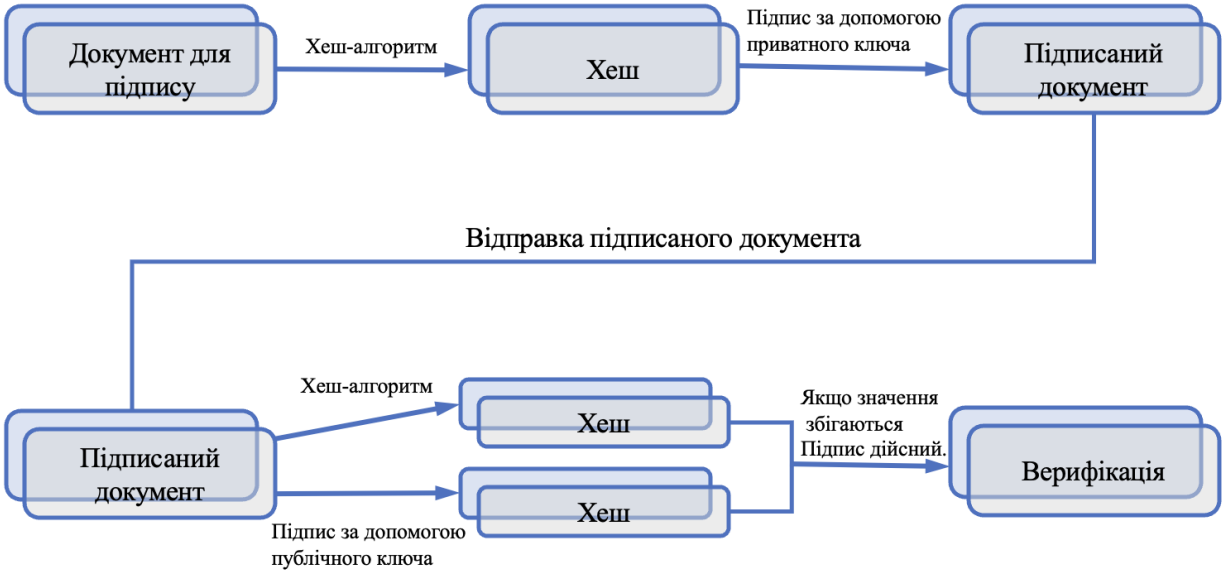

Рисунок 1.12 – Алгоритм створення та перевірки підпису

1.6.2 ECDSA

Згідно NIST наведено три основні функції для реалізації ECDSA це генерація ключів (більш детально розглянуто у розділі 1.4.2), створення цифрового підпису та його верифікація [22].

Генерація пари ключів

Вхідні дані: дійсний набір параметрів області еліптичної кривої.

Вихід: пара ключів (Q, d), пов’язана з параметрами області еліптичної кривої.

1. Вибираємо статистично унікальне і непередбачуване ціле число d в інтервалі $[1, n-1]$. Допускається використання випадкового чи псевдовипадкового числа. Якщо використовується метод псевдовипадкової генерації, початкові значення, які використовуються для генерації d, можуть визначатися внутрішніми засобами, бути

наданими абонентом або обома — це вибір реалізації. У всіх випадках вихідні значення мають ті самі вимоги безпеки, що й значення закритого ключа. Тобто вони мають бути захищені від несанкціонованого розголошення та бути непередбачуваними [22, 23].

2. Обчислюємо точку $Q = (xg, yQ) = dG$.

3. Пара ключів (Q, d), де Q — відкритий ключ, а d — закритий ключ.

Генерація цифрового підпису

Для того, щоб підписати деяке повідомлення m відправник А повинен зробити наступне [22, 24]:

1) Обрати випадкове ціле число $k \in [1,\ n-1]$.

2) Обчислити $k * g = (x, y)$ та $r = x\ mod\ n$. Якщо $r \equiv 0\ mod\ n$, то обирають нове випадкове число $k \in [1,\ n-1]$.

3) Обчислити $k^{-1}\ mod\ n$ та $s = k^{-1}\ (hash + d * r)\ mod\ n$, де hash – значення геш-функції повідомлення m, що підписується. Якщо $s = 0$, то значення $s^{-1}\ mod\ n$ не існує, тому необхідно повернутись до п.1.

Результатом процесу підпису є два цілі числа r і s - цифровий підпис. Алгоритм генерації цифрового підпису зображений на рисунку 1.13.

Перевірка цифрового підпису

Для того, щоб перевірити підпис відправника А на повідомлення, отримувач В повинен зробити наступне [22, 24]:

1. Отримати копію відкритого ключа Q відправника А.

2. Перевірити, що числа r та s є цілими числами з інтервалу $[1,\ n-1]$ та обчислити значення геш-функції hash від повідомлення.

3. Обчислити $u_1 = s^{-1} hash\ mod\ n$ та $u_2 = s^{-1}\ r\ mod\ n$.

4. Обчислити $u_1 g + u_2 Q = (x, y)$.

5. Прийняти підпис, якщо $x\ mod\ n = r\ mod\ n$.

Алгоритм перевірки підпису зображений на рисунку 1.14.

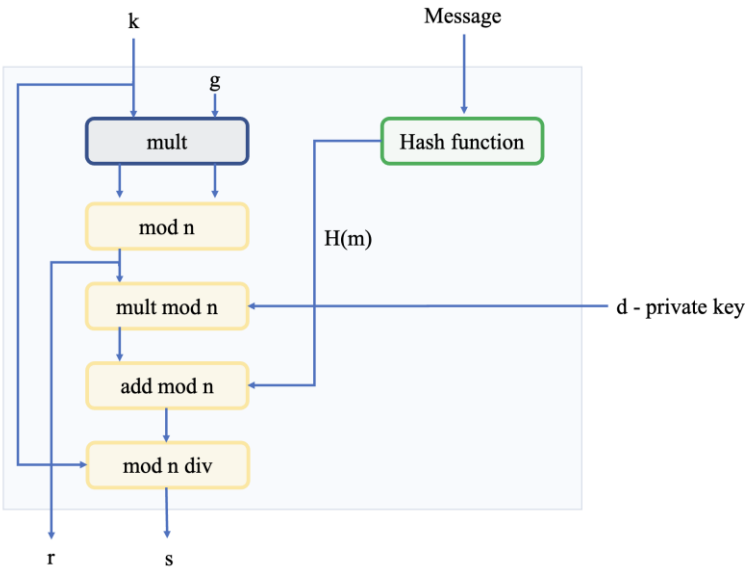

Рисунок 1.13 – Генерація підпису ECDSA

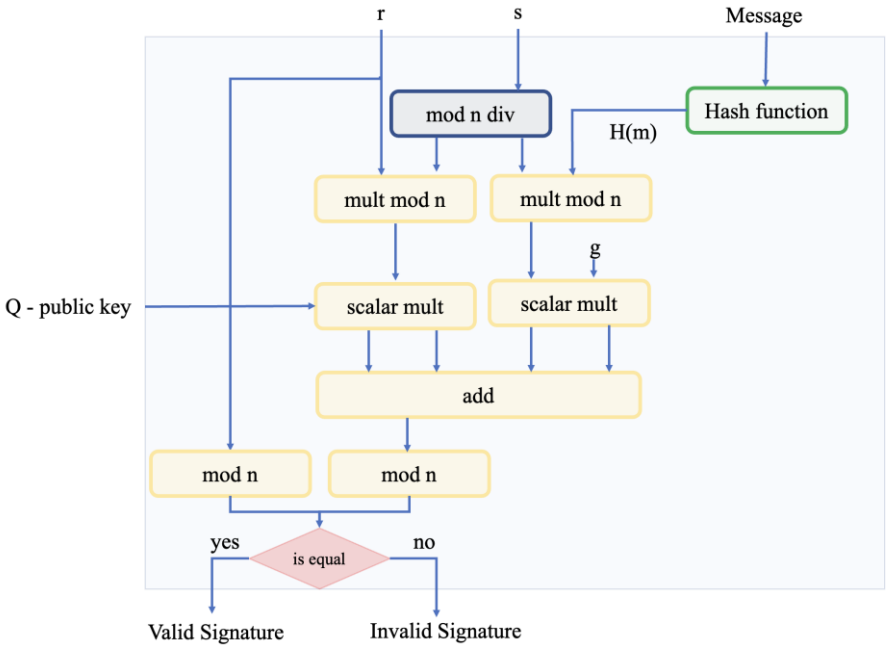

Рисунок 1.14 – Перевірка підпису ECDSA.

Можна легко показати, що даний алгоритм дійсно успішно засвідчує цифровий підпис. Якщо підпис (r, s) повідомлення m було дійсно згенеровано з використанням секретного ключа, то $s = k^{-1}\ (hash + d * r)\ mod\ n$. Розкриваючи дужки отримаємо:

$$k \equiv s^{-1}\ (hash + d * r) \equiv s^{-1} * hash + s^{-1} * d * r \equiv u_1 + u_2 * d\ mod\ n \quad (1.11)$$

Тоді $u_1 * g + u_2 * Q = (u_1 + u_2 * d) * g = kg$ і значить $x = r$, що і вимагається для підтвердження підпису.

1.7 Висновки

У даному розділі було розглянуто необхідність забезпечення безпечного обміну інформації шляхом криптографічних механізмів для забезпечення цілісності та конфіденційності даних. Порівнюючи блокові алгоритми шифрування можна побачити, що алгоритми AES та Blowfish переважно кращі за інші алгоритми в досліджені наприклад DES та 3-DES програють по рівню безпеки, IDEA в свою чергу, має низький рівень гнучкості та стійкості до атак. Blowfish має досить високі показники з безпеки та продуктивності, але час на розшифрування та стійкість до криптоаналізу досить низькі. AES є універсальним рішенням та з урахуванням всіх критеріїв переваги компенсують недоліки. Тому у подальшому досліджується саме AES. Як спосіб забезпечити більшу криптостійкість ключів AES, було обрано алгоритм генерації ключів на основі еліптичних кривих. У даному розділі описані основні аспекти алгоритму ECC.

Криптографічні засоби захисту інформації, як кваліфіковані електроні підписи, є невід'ємною складовою захисту конфіденційної інформації. Еліптичні криві, як базовий математичний об'єкт для створення асиметричних криптографічних алгоритмів, займають все більш важливе місце у криптографії. Алгоритм ECDSA є безпечнішим в порівнянні з RSA завдяки своїй складності. ECDSA забезпечує такий же рівень безпеки, як і RSA, але використовує меншу довжину ключа. Тож ECDSA більш безпечний криптографічний алгоритм, який використовується як для кваліфікованого цифрового підпису, його особливості були розглянуті в даному розділі.

РОЗДІЛ 2
АРХІТЕКТУРА ПРОГРАМНОГО ЗАСОБУ

2. Опис архітектури програмного засобу

2.1 Огляд вимог та функціональності програмного забезпечення

Цілю розробки програмного забезпечення є створення програмного засобу, що забезпечує цілісність та конфіденційність даних за допомогою використання блочного алгоритму шифрування AES та електронного цифрового підпису ECDSA на основі еліптичних кривих. Для досягнення цієї мети були сформульовані наступні вимоги та функціональність програмного забезпечення:

1. Функція менеджера сертифікатів:
   - Можливість імпортування та експортування сертифікатів.
   - Можливість керування сертифікатами
   - Можливість поділитися сертифікатом через пошту.
2. Шифрування та розшифрування файлів і тексту:
   - Можливість шифрування/розшифрування вмісту файлів за допомогою алгоритму AES.
   - Можливість шифрування/розшифрування тексту у наявному notepad.
3. Цифровий підпис:
   - Можливість генерації електронного цифрового підпису для документів на основі алгоритму ECDSA.
   - Можливість перевірки цифрового підпису для підтвердження цілісності документів.
4. Управління ключами:
   - Можливість генерації ключів шифрування та підпису на основі еліптичних кривих.
   - Можливість зберігання та керування ключами шифрування та підпису.

5. Безпека:

- Захист сертифікатів та ключів шифрування/підпису за допомогою шифрування з використанням AES при зберіганні та імпортуванні.
- Забезпечення захищеного зберігання сертифікатів локально на комп’ютері користувача.
- Без успішної авторизації користувача не має доступу до сертифікатів та програмного забезпечення.

6. Користувацький інтерфейс:

- Зручний та інтуїтивно зрозумілий інтерфейс для взаємодії з програмним забезпеченням.
- Можливість легкого навігації та виконання функцій програми.

7. Підтримка платформ:

- Робота програмного забезпечення на операційних системах Linux, Windows та MacOS.

Виконання цих вимог та функціональності дозволить забезпечити ефективну та безпечну роботу з даними, збереження їх конфіденційності та цілісності, а також забезпечить зручне та інтуїтивно зрозуміле користувацьке взаємодію з програмним забезпеченням.

2.2 Взаємодія компонентів

Взаємодія модулів програмного засобу зображено на рисунку 2.1. Після успішної авторизації користувач переходить до головного вікна програми, де представлені наступні функціоналі блоки: менеджер сертифікатів, управління акаунтами, взаємодія з файлами та взаємодія з текстом. Менеджер сертифікатів складається з таких частин як: створення сертифікату, імпорт, експорт та можливість поділитися сертифікатом поштою. Взаємодія з файлами включає в себе шифрування/розшифрування, підпис та перевірка. Взаємодія з текстом представляє шифрування та розшифрування тексту за допомогою вбудованого блокноту.

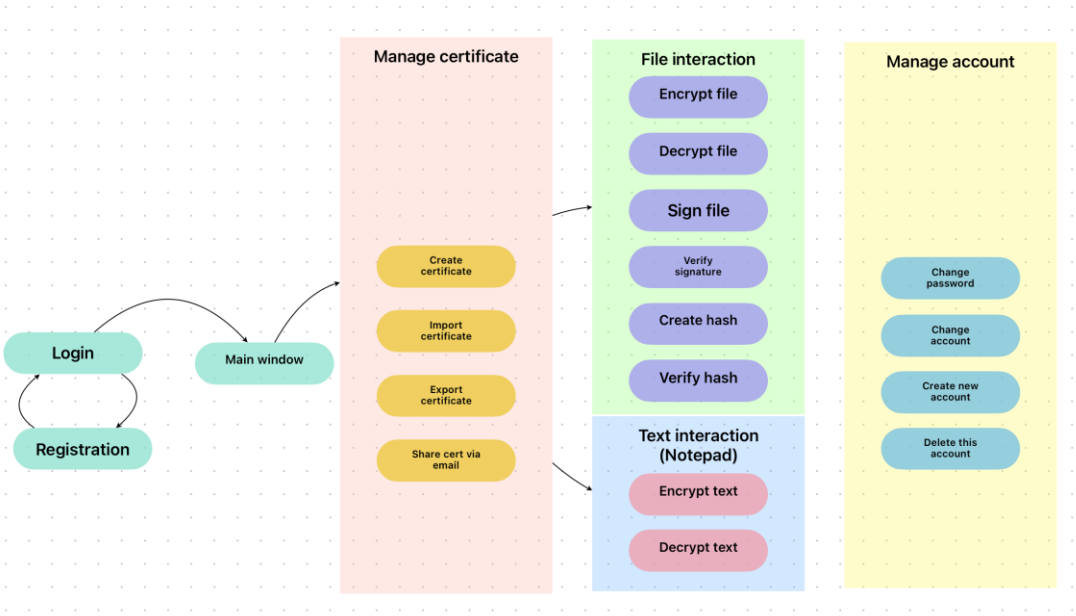

Рисунок 2.1 – Функціональні модулі програмного засобу

2.3 ECC AES

Для того, щоб зашифрувати повідомлення, використовуючи алгоритм AES та генерацію ключів ECC, треба створити сертифікат використовуючи програмний модуль «Create certificate», після чого отримуємо пару публічний ключ (що складається з двох координат x, y) та приватний. Використовуючи відкритий ключ, виконується шифрування та розшифрування повідомлення. Алгоритм зображений на рисунку 2.2.

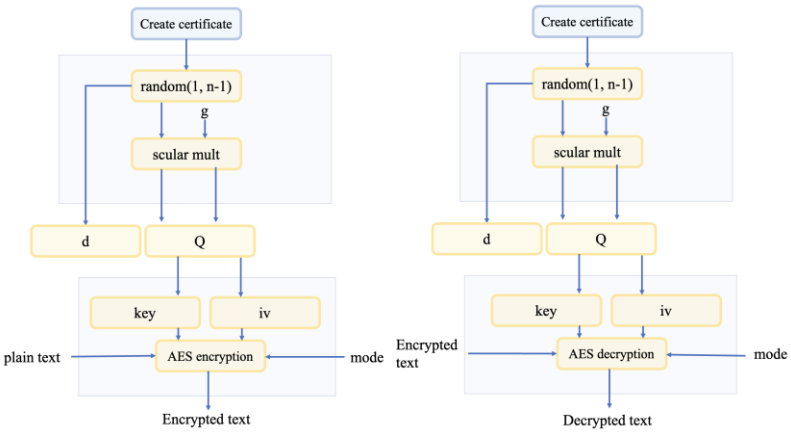

Рисунок 2.2 – Комбінація алгоритмів ECC та AES для шифрування та розшифрування

2.3.1 Еліптичні криві

Згідно SEC рекомендовані параметри області еліптичної кривої над F_p отримали псевдоніми, щоб їх було легко ідентифікувати. Назви вибирали такі, що кожна назва починається з sec для позначення «Стандартів ефективної криптографії», за якою йде p для позначення параметрів над F_p, після чого йде цифра, що позначає довжину поля розміру p у бітах, за якою йде k для позначення параметрів, пов'язаних із кривою Кобліца [25].

Отже, були виділені рекомендовані еліптичні криві secp192k1, secp224k1, secp256k1 [26].

Параметри secp192k1 зображені на рисунку 2.3.

```
ELLIPTIC_CURVE_192k1 = EllipticCurve(
    'secp192k1',
    p=0xfffffffffffffffffffffffffffffffffffffffeffffee37,
    a=0,
    b=3,
    g=(0xdb4ff10ec057e9ae26b07d0280b7f4341da5d1b1eae06c7d,
       0x9b2f2f6d9c5628a7844163d015be86344082aa88d95e2f9d),
    n=0xfffffffffffffffffffffe26f2fc170f69466a74defd8d,
    h=1,
)
```

Рисунок 2.3 – Параметри еліптичної кривої secp192k1

Параметри secp224k1 зображені на рисунку 2.4.

```
ELLIPTIC_CURVE_224k1 = EllipticCurve(
    'secp224k1',
    p=0xffffffffffffffffffffffffffffffffffffffffffffffffeffffe56d,
    a=0,
    b=5,
    g=(0xa1455b334df099df30fc28a169a467e9e47075a90f7e650eb6b7a45c,
       0x7e089fed7fba344282cafbd6f7e319f7c0b0bd59e2ca4bdb556d61a5),
    n=0x010000000000000000000000001dce8d2ec6184caf0a971769fb1f7,
    h=1,
)
```

Рисунок 2.4 – Параметри еліптичної кривої secp224k1

Параметри secp256k1 зображені на рисунку 2.5.

```
DEFAULT_ELLIPTIC_CURVE = EllipticCurve(
    'secp256k1',
    p=0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f,
    a=0,
    b=7,
    g=(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,
       0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8),
    n=0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141,
    h=1,
)
```

Рисунок 2.5 – Параметри еліптичної кривої secp256k1

2.3.2 Реалізація алгоритму створення ключів

Реалізація алгоритму визначає клас *ECCKeyGenerator* для генерації криптографічного ключа еліптичної кривої. Клас містить методи для генерації пари відкритий-приватний ключ, перевірки наявності точки на кривій, заперечення точки, додавання двох точок і скалярного множення точки на скаляр [27]. Опис кожного методу [28]:

- *__init__(self, private_key=None, curve: EllipticCurve = DEFAULT_ELLIPTIC_CURVE):* ініціалізує новий екземпляр класу *ECCKeyGenerator* із додатковим закритим ключем та еліптичною кривою. Якщо приватний ключ не надано, генерується випадкове число в діапазоні порядку кривої.
- *gen_keypair(self):* генерує пару відкритий-приватний ключ, використовуючи закритий ключ екземпляра та точку генератора кривої.
- *extended_gcd(k, p):* реалізує розширений алгоритм Евкліда для обчислення найбільшого спільного дільника k і p, а також модульного оберненого k відносно p. Це використовується в методах додавання та скалярного множення.
- *is_on_curve(self, point):* Визначає, чи є точка на еліптичній кривій.
- *point_neg(self, point):* Заперечує точку на кривій шляхом зміни знака координати y.
- *add_point(self, point1, point2):* додає дві точки на кривій за допомогою формул додавання еліптичної кривої. Якщо будь-яка точка має значення None, повертає іншу точку. Якщо координати x точок однакові, але їхні координати y різні,

повертає None. В іншому випадку обчислює нахил лінії між точками, обчислює координати x і y точки суми та повертає їх.

- *scalar_mult(self, k, point):* виконує скалярне множення на точку на кривій шляхом багаторазового додавання точки до себе k разів за допомогою двійкового розширення k. Метод приймає два аргументи: k, цілий скаляр, і point, кортеж, що представляє точку на еліптичній кривій. Метод спочатку перевіряє, чи знаходиться точка на кривій за допомогою методу *is_on_curve()*. Якщо скаляр дорівнює нулю або точка None, він повертає None. Якщо скаляр від'ємний, метод рекурсивно викликає себе з абсолютним значенням скаляра та запереченням точки. Це еквівалентно зміні напрямку множення на кривій. Створений результат входить у цикл, який виконує двійкове розкладання скаляра. У кожній ітерації перевіряється молодший біт скаляра, і якщо він дорівнює 1, додається до результату за допомогою методу *add_point()*. Потім він подвоює доданий, додаючи його сам до себе, і зсуває біти скаляра вправо. Цей процес повторюється, поки скаляр не стане нульовим. Результат становить точку, яка є відкритим ключем.

Повний програмний код можна переглянути у додатку 1.

2.3.3 Реалізація алгоритму AES

На даний момент є безліч реалізацій алгоритму AES на мові програмування Python, також за допомогою існуючих модулів. Саме такий модуль було використано для реалізацію шифрування та розшифрування. На наведеному рисунку 2.6 зображений клас, який реалізовує шифрування та розшифрування при введені вхідний даних такі як, ключ, вектор ініціалізації та режим роботи (даний програмний код універсальний для CBC, CFB, OFB режимів роботи) [28].

При шифруванні створюємо новий об'єкт AES, використовуючи значення ключа, режиму та ініціалізаційного вектору як аргументи функції, кодуємо вхідні дані та використовує створений об'єкт AES, щоб здійснити шифрування вхідних даних. Крім того, за допомогою функції pad до даних додається необхідна кількість байтів, щоб забезпечити кратність блоку розміру AES (16 байтів). Нарешті, результат

шифрування та ініціалізаційний вектор об'єднуються, додаються та кодуються у форматі Base64, щоб створити закодований текст, який може бути використаний для безпечної передачі та збереження даних. Як результат отримуємо шифротекст.

aes_copy.py > AESCipher

```python
from base64 import b64decode
from base64 import b64encode

from Cryptodome.Cipher import AES
from Cryptodome.Util.Padding import pad, unpad


class AESCipher:
    def __init__(self, key, iv, mode):
        self.key = key.encode('utf-8')
        self.iv =  iv.encode('utf-8')
        self.mode = getattr(AES, f"MODE_{mode}")


    def encrypt(self, data):
        self.cipher = AES.new(self.key, self.mode, self.iv)
        data = data.encode('utf-8')
        return b64encode(self.iv + self.cipher.encrypt(pad(data,
            AES.block_size)))


    def decrypt(self, data):
        raw = b64decode(data)
        self.cipher = AES.new(self.key, self.mode, raw[:AES.block_size])
        return unpad(self.cipher.decrypt(raw[AES.block_size:]), AES.block_size)

```

Рисунок 2.6 – програмна реалізація алгоритму AES

При розшифрувані декодуємо шифровані дані, так само створюємо новий об'єкт AES, використовуючи значення ключа, режиму та вектору ініціалізації, який отримано з розшифрованої частини закодованих даних розміром AES.block_size (16 байтів), який використовувався як вектор ініціалізації при шифруванні. Далі розшифровуємо частину даних, за допомогою створеного об'єкту AES та використовуючи вказаний розмір блоку AES. Після розшифрування використовується функція unpad для видалення доданих байтів, що були додані під час шифрування. Нарешті, розшифрований текст повертається як результат виконання функції.

2.4 ECDSA

Для того, щоб підписати документ, використовуючи алгоритм ECDSA, треба створити сертифікат використовуючи програмний модуль «Create certificate», після чого отримуємо пару публічний ключ (що складається з двох координат x, y) та приватний. Використовуючи приватний ключ, виконується підпис документа, щоб перевірити підпис необхідний лише публічний ключ. Алгоритм зображений на рисунку 2.6.

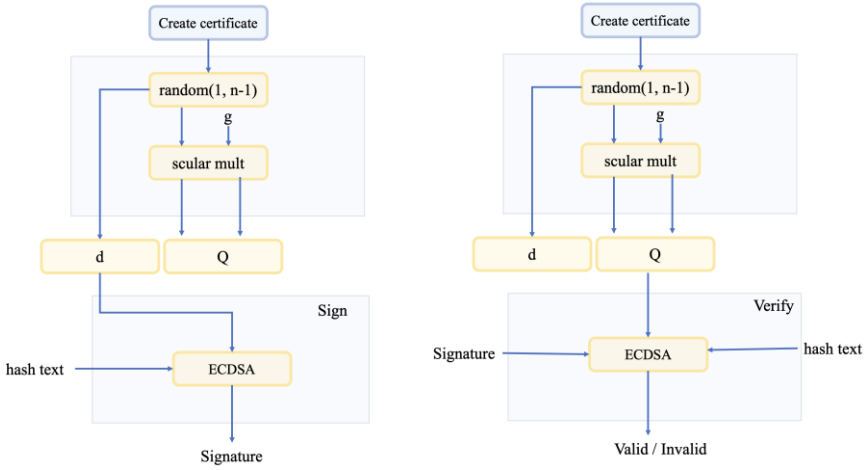

Рисунок 2.6 – Алгоритм створення та перевірки підпису ECDSA

2.4.1 Програмна реалізація ECDSA

Для реалізації ECDSA використовуємо три функції для отримання гешу повідомлення, створення підпису та верифікація. Ключі, які використовуються у даному коді, згенеровані раніше.

Функція *hash_message()* (рисунок 2.7) гешує повідомлення message за допомогою SHA-512 (Secure Hash Algorithm 512-bit), який повертає 64-байтний геш (512 бітів). Далі береться цей геш та перетворюється в ціле число e за допомогою функції *int.from_bytes()* з параметром 'big', що означає, що байти повинні бути прочитані у великому порядку байтів (найбільш значущі байти спочатку). Далі отримане число e зменшується за допомогою побітового здвигу на різницю між

бітовою довжиною числа e та бітовою довжиною параметру кривої curve.n (модуль кривої). Отримане число стає значенням гешування повідомлення [28].

```
def hash_message(message, curve):
    message_hash = hashlib.sha512(message).digest()
    e = int.from_bytes(message_hash, 'big')
    hash = e >> (e.bit_length() - curve.n.bit_length())
    assert hash.bit_length() <= curve.n.bit_length()
    return hash
```

Рисунок 2.7 – функція *hash_message()*

Програмний код функції *sign_message* (рисунок 2.8) реалізує процес підпису повідомлення за допомогою еліптичної криптографії. У першому рядку коду створюється об'єкт ecc для того щоб використовувати функції еліптичної криптографії та параметри кривої. Далі, з використанням функції *hash_messag*e(), повідомлення message гешується за допомогою SHA-512, що дає унікальне значення hash довжиною в n бітів (де n - порядок кривої Едвардса). У наступному кроці використовується цикл while, який продовжується доти, доки не будуть згенеровані значення r та s. Під час кожної ітерації генерується випадкове значення k, яке використовується для обчислення координат точки на кривій. За допомогою *scalar_mult* з об'єкта *ecc* обчислюється точка з координатами (x,y). Далі, r обчислюється як x модулю curve.n. S обчислюється як (*hash + r * int(private_key)) * ecc.extended_gcd(k, curve.n)) % curve.n*. В цій формулі *private_key* - приватний ключ, k - випадкове число з кожної ітерації циклу while. *extended_gcd()* - функція розширеного алгоритму Евкліда.

```
def sign_message(private_key, message, curve):
    ecc = ecdhe.ECCKeyGenerator()
    hash = hash_message(message, curve)
    r = 0
    s = 0
    while not r or not s:
        k = random.randrange(1, curve.n)
        x, y = ecc.scalar_mult(k, curve.g)
        r = x % curve.n
        s = ((hash + r * int(private_key)) * ecc.extended_gcd(k, curve.n)) % curve.n
    return r, s
```

Рисунок 2.8 – функція *sign_message()*

Отже, на виході отримуємо значення r та s, які разом із повідомленням можуть бути переслані отримувачу, який може перевірити підпис за допомогою публічного ключа [28].

Функція *verify_signature()* перевіряє підпис повідомлення за допомогою криптографії еліптичної кривої. *Public_key* – це точка на еліптичній кривій, *message* – це повідомлення, яке перевіряється, *signature* – це підпис повідомлення, а *curve* – це еліптична крива, яка використовується. Спочатку функція обчислює геш повідомлення за допомогою функції *hash_message*(). Потім отримуємо значення r і s із підпису. Обчислюємо w, що є мультиплікативним оберненим до s за модулем curve.n. Також розраховуємо u1 і u2, які є проміжними значеннями, які використовуються в перевірочному обчисленні. Далі використовуємо функції *scalar_mult* і *add_point* з класу *ECCKeyGenerator* для обчислення точки x і y на еліптичній кривій. Ці значення виводяться з відкритого ключа curve.g і проміжних значень u1 і u2. Функція порівнює значення r із сигнатури зі значенням x, отриманим у результаті обчислення. Якщо вони збігаються, функція повертає «підписи збігаються», вказуючи, що підпис дійсний. В іншому випадку функція повертає «недійсний підпис».

```
def verify_signature(public_key, message, signature, curve):
    ecc = ecdhe.ECCKeyGenerator()
    hash = hash_message(message, curve)
    r, s = signature
    w = ecc.extended_gcd(s, curve.n)
    u1 = (hash * w) % curve.n
    u2 = (r * w) % curve.n
    x, y = ecc.add_point(ecc.scalar_mult(u1, curve.g),
                         ecc.scalar_mult(u2, public_key))
    if (r % curve.n) == (x % curve.n):
        return 'signature matches'
    return 'invalid signature'
```

Рисунок 2.9 – Функція *verify_signature()*

2.5 Висновки

У даному розділі було розглянуті вимоги до програмного забезпечення, функціональні модулі, взаємодію основних модулів. Також було розглянуті програмні реалізації алгоритмів, що використовуються в програмному забезпеченні, були розриті можливості комбінацій криптографічних методів для забезпечення відповідного рівня безпеки. Оскільки алгоритм шифрування AES є симетричним, його криптостійкість напряму залежить від складності ключа, тож у даній роботі було використання поєднання AES з ECC.

РОЗДІЛ 3
ПРОГРАМНИЙ ЗАСТОСУНОК ДЛЯ ЗАБЕЗПЕЧЕННЯ ЦІЛІСНОСТІ ТА КОНФІДЕНЦІЙНОСТІ ІНФОРМАЦІЇ

3. Опис програмного застосунку

У даному розділі виділені основні функціональні можливості розробленого застосунку під назвою ELLICE, починаючи від авторизації, управління акаунтом та сертифікатами закінчуючи шифруванням, електронним цифровим підписом та безпекою даних у застосунку. Дане програмне забезпечення може бути використане як для особистого спілкування так і для внутрішньо-корпоративного документообігу. ELLICE є переважно локальною, тобто вся інформація щодо користувача та сертифікати, які створюються у програмі зберігаються лише на локальній системі, на якій були створені. Щодо процесу обміну інформацією або сертифікатами користувач може скористатися функціями які передбачають поширення сертифікатів за допомогою пошти або завантажити сертифікат на систему після чого поділитися їм зручним для користувача методом [28, 29].

3.1 Авторизація та реєстрація

При запуску програми перше, що бачить користувач це вікно авторизації у застосунку. Вікно авторизації можемо побачити на рисунку 3.1 . Якщо користувач має акаунт у системі, тобто був попередньо зареєстрований, то для того щоб увійти в акаунт користувачу треба ввести логін та пароль від акаунта. Логін як правило використовується як email, а пароль в свою чергу користувач створює самостійно, але бажано, щоб пароль був більше 8 символів з різним реєстром літер та спеціальними знаками, щоб забезпечити більший захист від взлому.

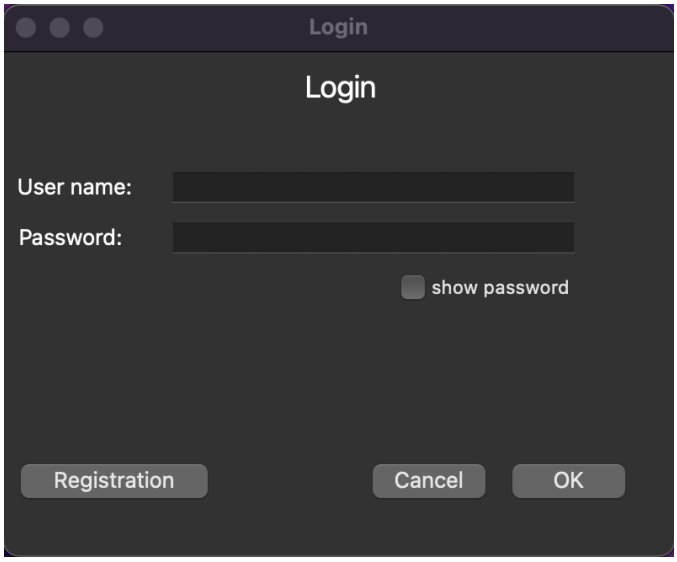

Рисунок 3.1 – Вікно авторизації

Також на початковому вікні є кнопка реєстрації, що переводить нас на реєстрацію нового акаунта, зображена на рисунку 3.2.

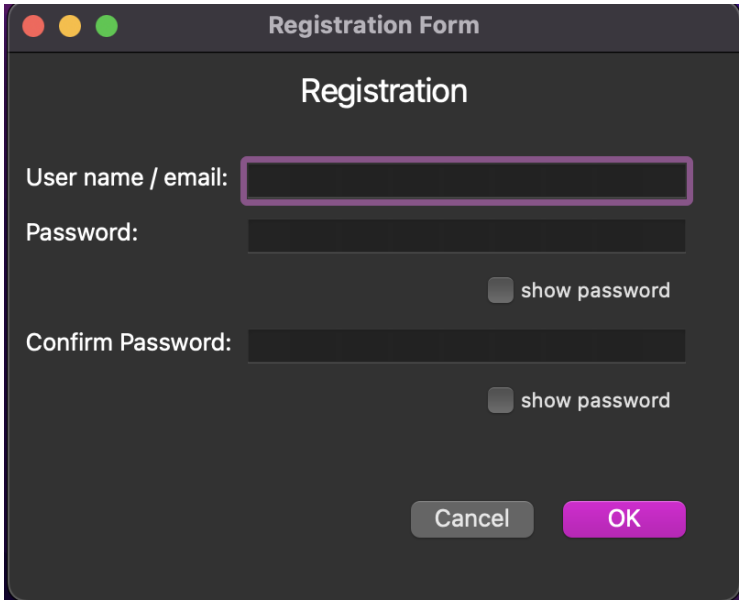

Рисунок 3.2 – Вікно реєстрації

При реєстрації користувач має заповнити своє ім’я або email, відповідно пароль та підтвердити його. Якщо паролі співпадають та такого користувача в системі не має, створюється новий акаунт. Створення нового акаунта відбувається так:

- Створюється файл в директорії програмного забезпечення як правило за шляхом `{os.path.expanduser('~')}/.ELLICE/` з назвою імені користувача в форматі .csv
- В файл записуються назви полів для подальшого зберігання сертифікатів користувача
- файл зашифровується паролем користувача використовуючи алгоритм шифрування AES, який описаний в розділі 1.3, 3.4.

Після проходження реєстрації користувача повертає на форму з авторизацією, де він має ввести свої облікові дані, після успішної авторизації користувач переходить до основного вікна програми, перед цим файл для зберігання сертифікатів розшифровується паролем та у головному вікні виводить інформацію про сертифікати, якщо вони є. Але якщо пароль користувача не підходить, для розшифрування файлу, то доступу до основного вікна програми не буде, а користувач побачить повідомлення щодо неправильного пароля (рисунок 3.3).

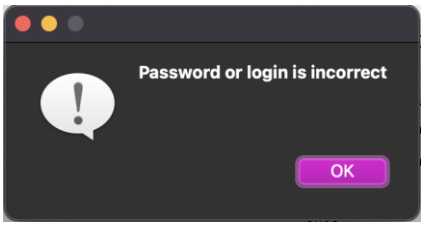

Рисунок 3.3 – Повідомлення про введення неправильного пароля

Таке саме повідомлення побачить користувач, який намагається авторизуватися з неіснуючим або неправильним логіном, це перевіряється шляхом перевірки на існування в відповідній папці файла з вказаним логіном, якщо ні то користувач отримає повідомлення та відповідно повернеться у вікно авторизації, де може повторити спробу або зареєструватися, якщо акаунт відсутній.

3.2 Створення сертифікатів

Після того як користувач успішно пройшов авторизацію він потрапляє у головне вікно програми, яке зображено на рисунку 3.3.

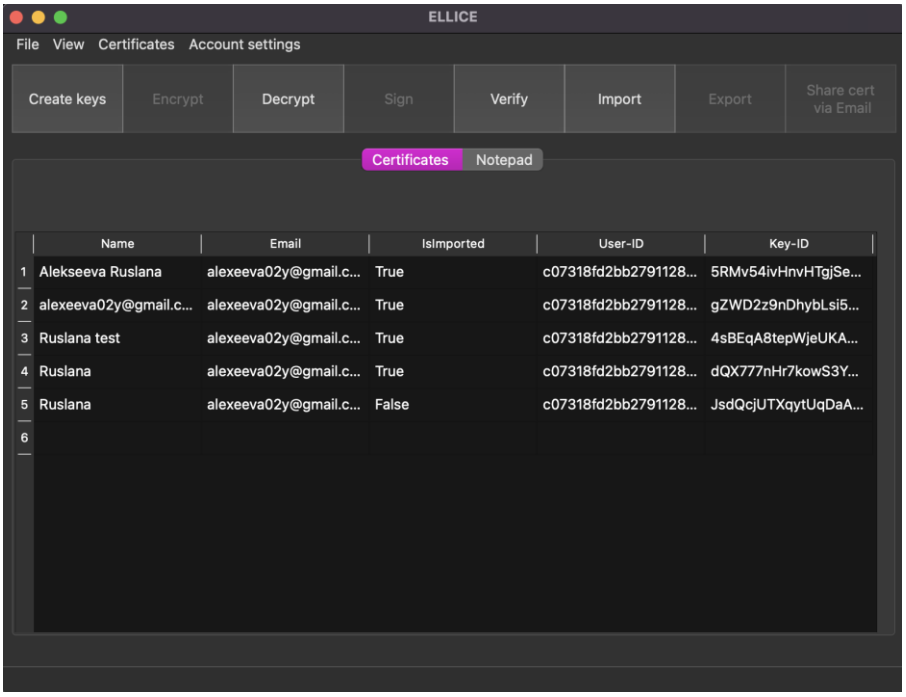

Рисунок 3.3 – Головне вікно програми

У вкладці сертифікатів при першій взаємодії з програмою не буде. Для того щоб створити новий сертифікат треба натиснути на кнопку «Create keys» після чого з’явиться вікно, яке зображено на рисунку 3.4.

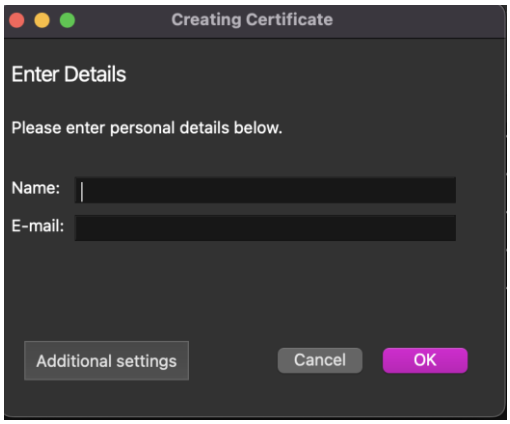

Рисунок 3.4 – Вікно створення сертифікату

У даному вікні створенні сертифікату треба ввести обов’язкові дані такі як ім’я та поштову адресу. Також є кнопка для додаткових налаштувань рисунок 3.5, де можна обрати еліптичну криву для створення ключів та відповідно режим роботи AES для шифрування (рисунки 3.6 та 3.7).

Якщо не змінювати додаткові налаштування, то за замовчуванням крива буде secp265k1, а режим роботи AES – CBC.

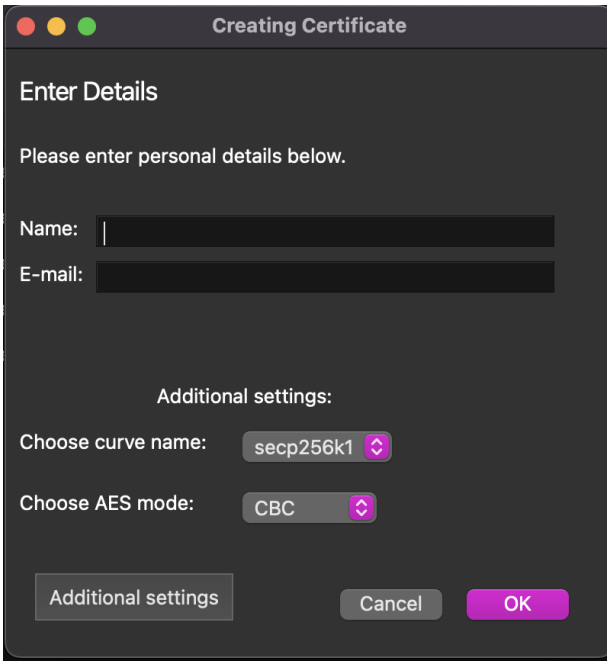

Рисунок 3.5 – Додаткові налаштування сертифікату

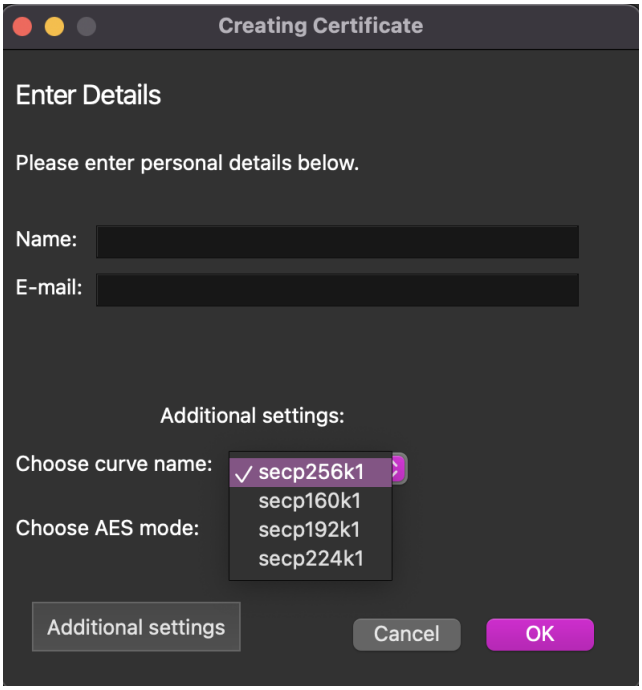

Рисунок 3.6 – Можливість обрати криву, за допомогою якої буде генеруватися сертифікат

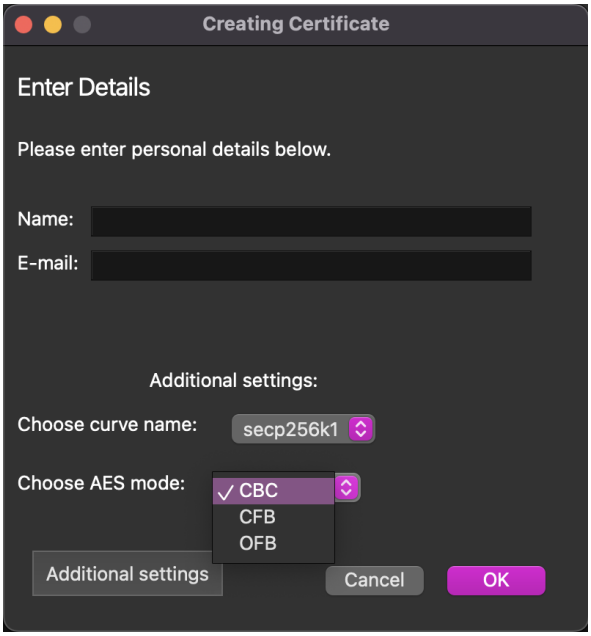

Рисунок 3.7 – Можливість обрати режим роботи AES

Після створення сертифікату новостворений сертифікат додається в табличку Рисунок 3.8, де є п’ять полів такі як: ім’я користувача, що було введено під час створення сертифікату; email, що був введений користувачем; чи був сертифікат імпортований в застосунок чи створений поточним користувачем; ідентифікатор користувача, що формується шляхом гешування email; ідентифікатор ключа – це унікальне значення для кожного сертифікату.

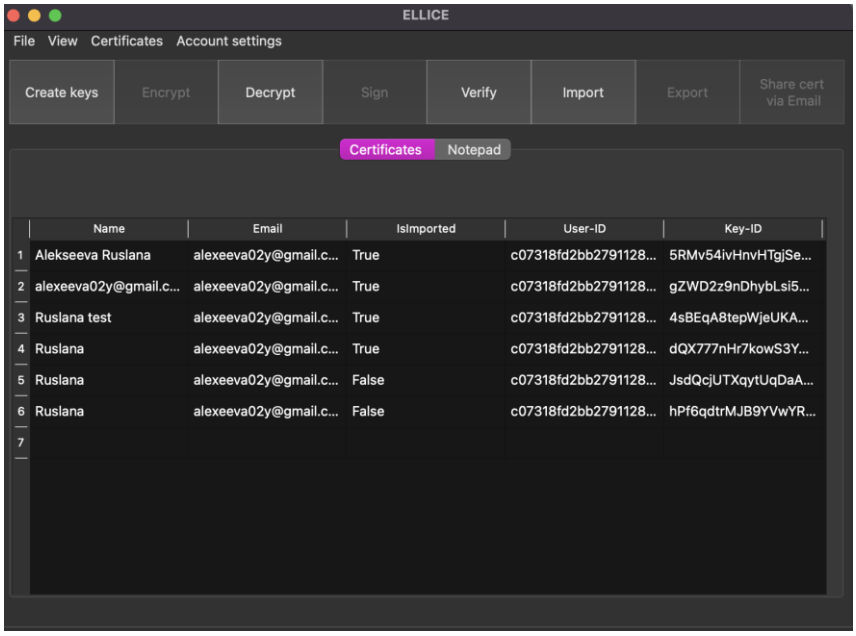

Рисунок 3.8 – Головне вікно програми, табличка з сертифікатами

Також можна в застосунку подивитися більш детальну інформацію, щодо сертифікату, яка двічі натиснути на обраний сертифікат рисунок 3.9.

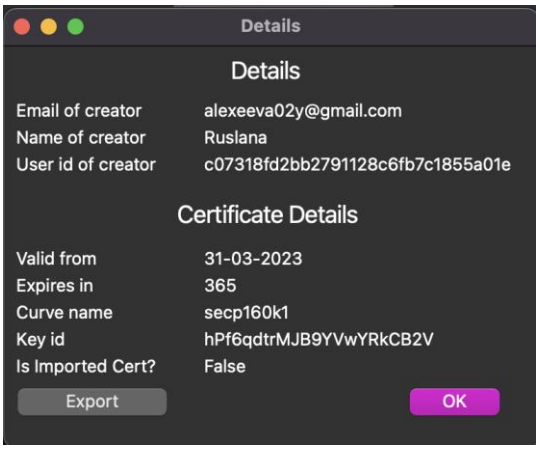

Рисунок 3.9 – Деталі обраного сертифікату

При перегляді деталей сертифікату додається інформація щодо дати кінцевої дії сертифікату, сертифікат в свою чергу дійсний рік, також є додаткова інформація щодо назви еліптичної кривої. Можна побачити, що є кнопка для Експорту сертифікату, що буде розглянуто в розділі 3.6.

3.3 Система зберігання сертифікатів

База даних для збереження сертифікатів користувача як було описано в Розділі 3.1, створюється під час реєстрації. Файл з сертифікатами завжди зашифрований і зберігається в директорії програми. Коли користувач вводить правильний пароль, програма не записує його в файл, а зберігає для поточного сеансу у змінній. Якщо користувач створює новий сертифікат або взаємодіє з існуючим, робиться запит в базу даних даного користувача і тільки з правильним паролем вдасться отримати інформацію або сертифікати. Варто зазначити, що файл розшифровується лише в програмі, відповідний файл у файловій системі завжди зашифрований.

Інформація, яка зберігається у файлі або базі даних користувача:

- user_id – геш email користувача
- name – ім’я користувача
- email – пошта користувача

- key_id – унікальне значення, яке ідентифікує ключі
- curve_name – назва кривої, за допомогою якої створили ключі
- mod – режим роботи, який обрав користувач для подальшого шифрування
- isimported – значення чи імпортований сертифікат чи ні (імпортований сертифікат не має приватного ключа)
- created_at – дата створення сертифікату
- private_key – приватний ключ, є якщо сертифікат створений поточним користувачем та не був імпортований
- public_key_x – перший публічний ключ (пара відкритих ключів x та y, які створені на основі еліптичної криптографії)
- public_key_y – другий публічний ключ (пара відкритих ключів x та y, які створені на основі еліптичної криптографії)

На рисунку 3.10 зображена таблиця з сертифікатами у не зашифрованому вигляді. На рисунку 3.11 зображений файл з інформацією що і на рисунку 3.10, але у зашифрованому вигляді, так як виглядає увесь час.

user_id	name	email	key_id	curve_name	mod	isimported	created_at	private_key	public_key_x	public_key_y
c07318fd2bb2791128c6fb7	Alekseeva Ruslana	alexeeva02y@gmail.com	5RMv54ivHnvHTgjSe	secp256k1	ECB	TRUE	20-02-2023		583647461234311953 71	10332404061739455480033
c07318fd2bb2791128c6fb7	alexeeva02y@gmail.com	alexeeva02y@gmail.com	gZWD2z9nDhybLsi59	secp160k1	CBC	TRUE	23-02-2023		19493752633070885891	11519960334117645900510
c07318fd2bb2791128c6fb7	Ruslana test	alexeeva02y@gmail.com	4sBEqA8tepWjeUKAF	secp256k1	CBC	TRUE	03-03-2023		208464555918370272 13	82206676693323264857299
c07318fd2bb2791128c6fb7	Ruslana	alexeeva02y@gmail.com	dQX777nHr7kowS3Y2	secp256k1	CBC	TRUE	03-03-2023		847419162268769408 7	10726710436815450930 56
c07318fd2bb2791128c6fb7	Ruslana	alexeeva02y@gmail.com	JsdQcjUTXqytUqDaA	secp256k1	CBC	FALSE	31-03-2023	10045869594915102	136245389222321366 59	82129242741526857657461
c07318fd2bb2791128c6fb7	Ruslana	alexeeva02y@gmail.com	hPf6qdtrMJB9YVwYR	secp160k1	CFB	FALSE	31-03-2023	14000186846843538	105251562899944650 93	13910576301078505820577

Рисунок 3.10 – Таблиця сертифікатів в не зашифрованому вигляді

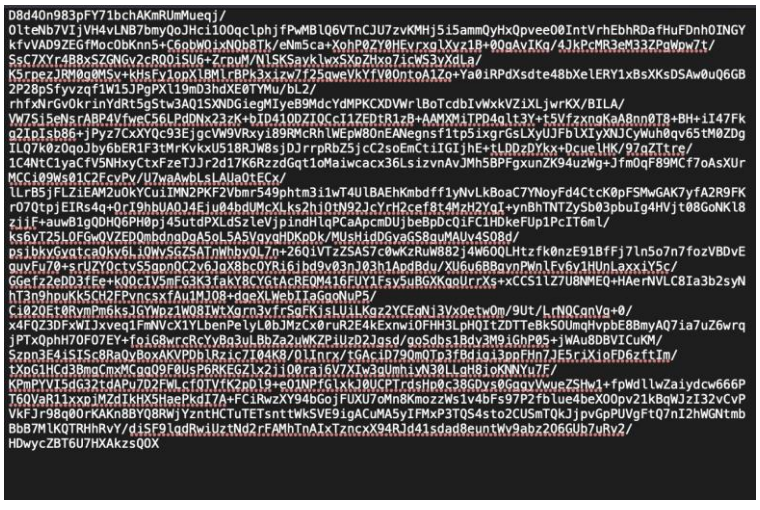

Рисунок 3.11 – Таблиця сертифікатів в зашифрованому вигляді

Тож тільки при взаємодії через програму ELLICE користувач може «редагувати» файл. Якщо користувач створює новий сертифікат для запису в базу, файл розшифровується в програмі, додається рядок про новий сертифікат та весь файл зашифровується паролем користувача. Так само і при інших функціях, які вимагають використання інформації з файлу. Якщо користувач закрив програму або змінив акаунт, то відповідно йому треба буде повторно проходити авторизацію.

Якщо у зловмисника немає пароля від акаунта шанс взлому та витоку інформації щодо сертифікатів мінімальний. При наявності пароля шанси більше, але все одно програмно інформація про ключі не надається.

3.4 Шифрування розшифрування файлів та тексту

Після того як користувач створив сертифікат, з’являється можливість шифрування даних. Щоб зашифрувати файл треба слідувати таким крокам:

1. Обрати в програмі створений раніше сертифікат та натиснути на нього в головному вікні (рисунок 3.12).
2. Після кроку 1 стає доступна кнопка «Encrypt» та натискаємо на неї.
3. Обраємо файл, який бажаємо зашифрувати (рисунок 3.13).
4. Отримаємо повідомлення про успішне зашифрування файлу (рисунок 3.14).

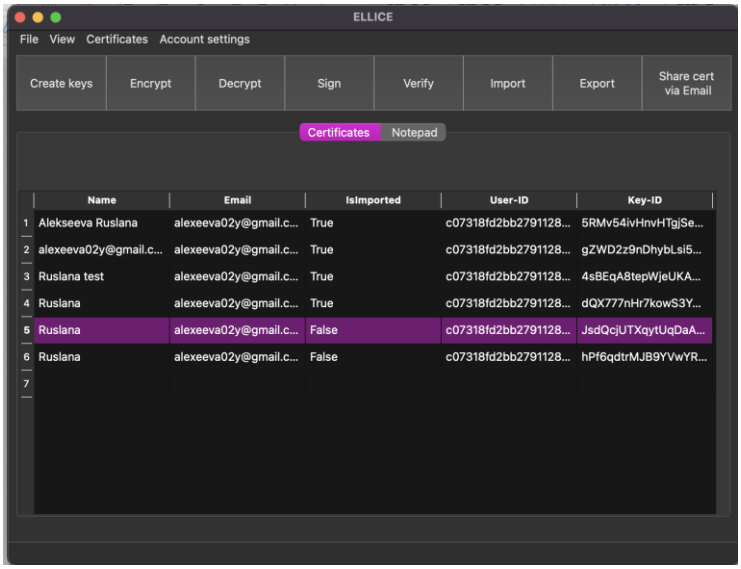

Рисунок 3.12 – Обраний сертифікат для шифрування

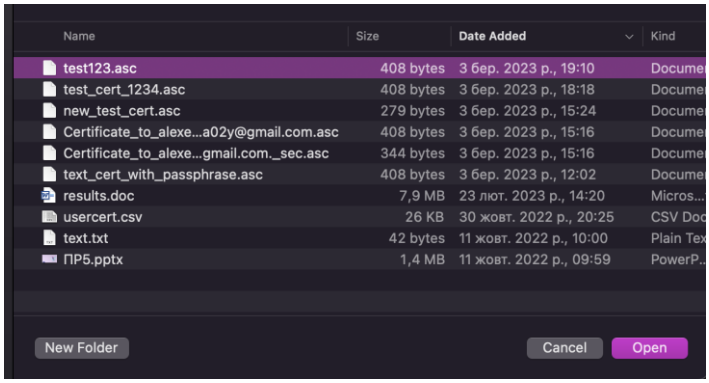

Рисунок 3.13 – Обираємо файл для шифрування

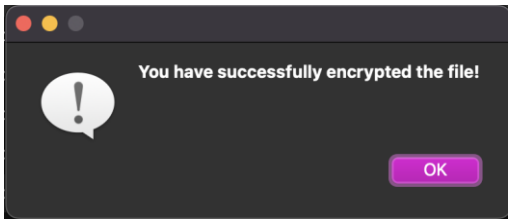

Рисунок 3.14 – Успішно зашифрований файл

Можемо переглянути файл який було зашифровано. Він виглядає як набір значень та символів. Файл який ми обрали для шифрування не видаляється та не змінюється. Програма лише створює такий самий файл, але з зашифрованою інформацією та додає в назву файлу «.enc» (що означає encrypted) тим самим змінивши його розширення (рисунок 3.15).

Рисунок 3.15 – Зашифрований файл

З технічної сторони шифрування виглядає так, при створені ключів отримуємо ключі та режим роботи відповідного сертифіката, обраний файл для шифрування відкриваємо для читання у бінарному форматі – це вхідний текст, далі викликаємо програму описану в розділі 2.3.3 та за допомогою бібліотеки AES і створеної функції encrypt зашифровуємо дані використовуючи відповідний режим роботи та публічні ключі як ключ та вектор ініціалізації алгоритму відповідно; далі отриманий шифротекст записується у новий файл і зберігається.

Розшифрування файлів виконується подібним чином як і шифрування:

1. Обраємо в програмі створений раніше сертифікат, яким був зашифрований файл та натискаємо на нього в головному вікні (рисунок 3.8).
2. Натискаємо на кнопку «Decrypt».
3. Обраємо файл, який бажаємо розшифрувати.
4. Отримаємо повідомлення про успішне розшифрування файлу.

Якщо було обрано не той сертифікат, яким було зашифровувано файл, отримаємо повідомлення про невдачу (рисунок 3.16).

Якщо отримали повідомлення про успіх, у поточну директорію, в якій знаходиться зашифрований документ, зберігається розшифрований файл.

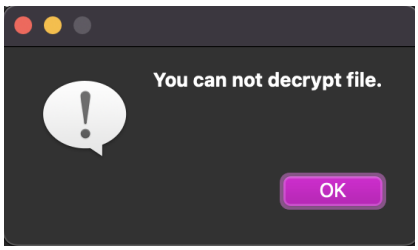

Рисунок 3.16 – Повідомлення про невдачу

З технічної сторони розшифрування виглядає так, при створені ключів отримуємо ключі та режим роботи відповідного сертифіката, обраний файл для розшифрування відкриваємо для читання у бінарному форматі – це шифротекст, далі викликаємо програму описану в розділі 2.3.3 та за допомогою бібліотеки AES і створеної функції decrypt розшифровуємо дані використовуючи відповідний режим роботи та публічні ключі як ключ та вектор ініціалізації алгоритму відповідно; далі отриманий розшифрований текст, який записується у новий файл без розширення «.enc» (тобто з такою назвою яка була перед шифруванням) і зберігається.

Також окрім файлів в програмі ELLICE можна шифрувати та розшифровувати текст у вбудованому notepad (рисунок 3.17). Notepad це додаткова вкладка для роботи з текстовим форматом для шифрування та розшифрування.

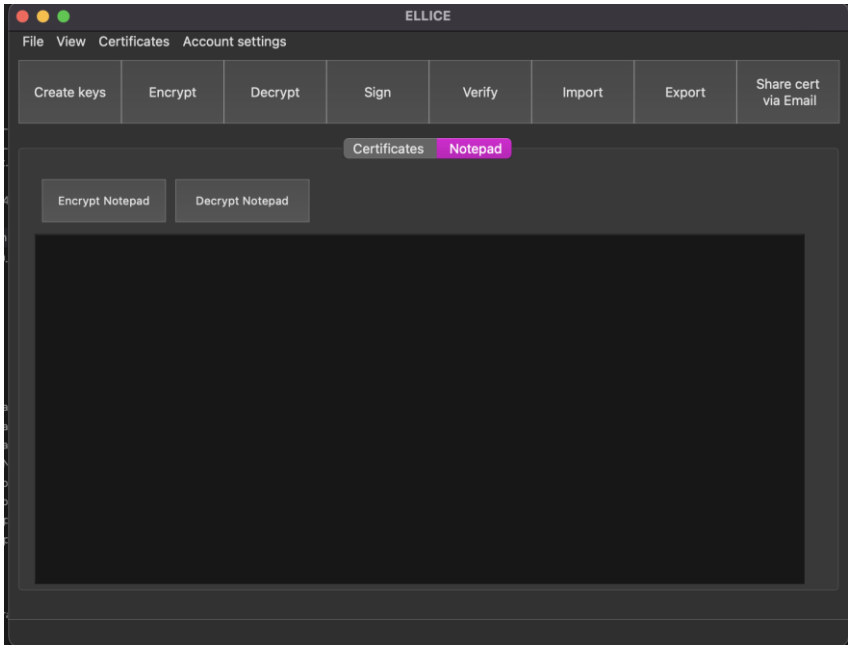

Рисунок 3.17 – Notepad

Процес аналогічний до шифрування та розшифрування за відмінністю, що результат шифрування та розшифрування можна зразу побачити на екрані. Так само попередньо обираємо сертифікат, вводимо текст та натискаємо на кнопку “Encrypt Notepad”. Після чого отримуємо шифротекст та при натисканні «Decrypt Notepad» отримуємо розшифрований текст (рисунки 3.18-3.20).

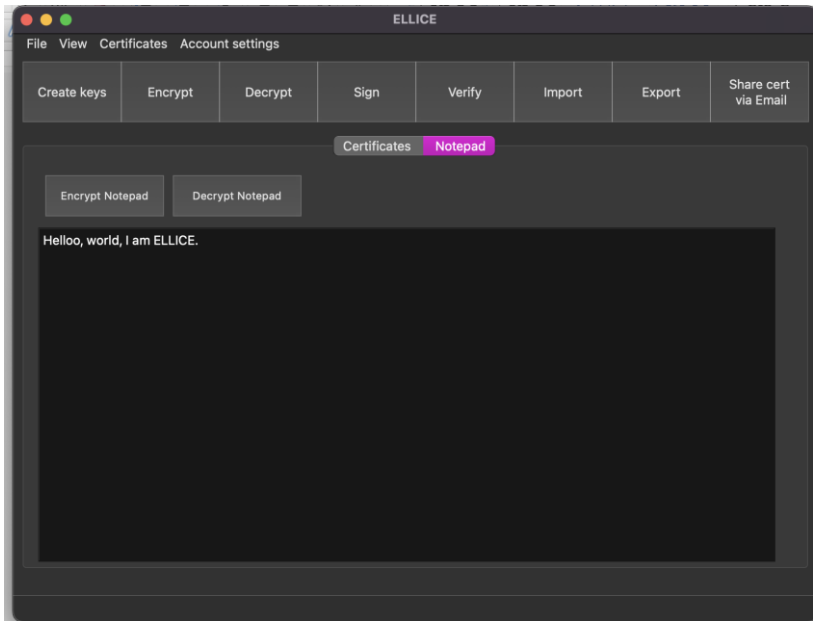

Рисунок 3.18 – Вводимо текст для шифрування

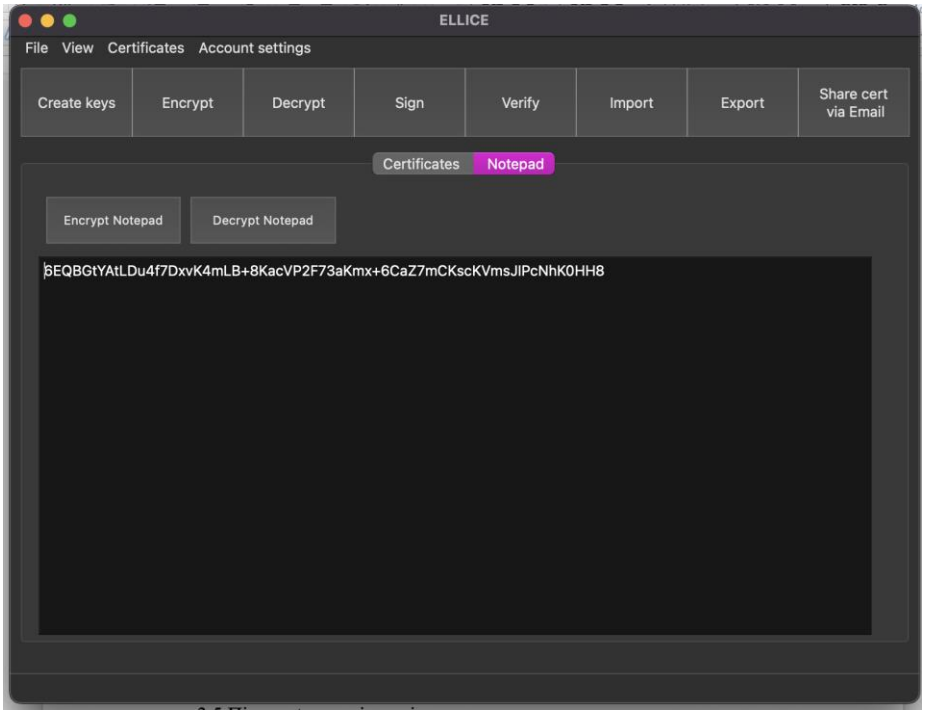

Рисунок 3.19 – Отримуємо шифротекст

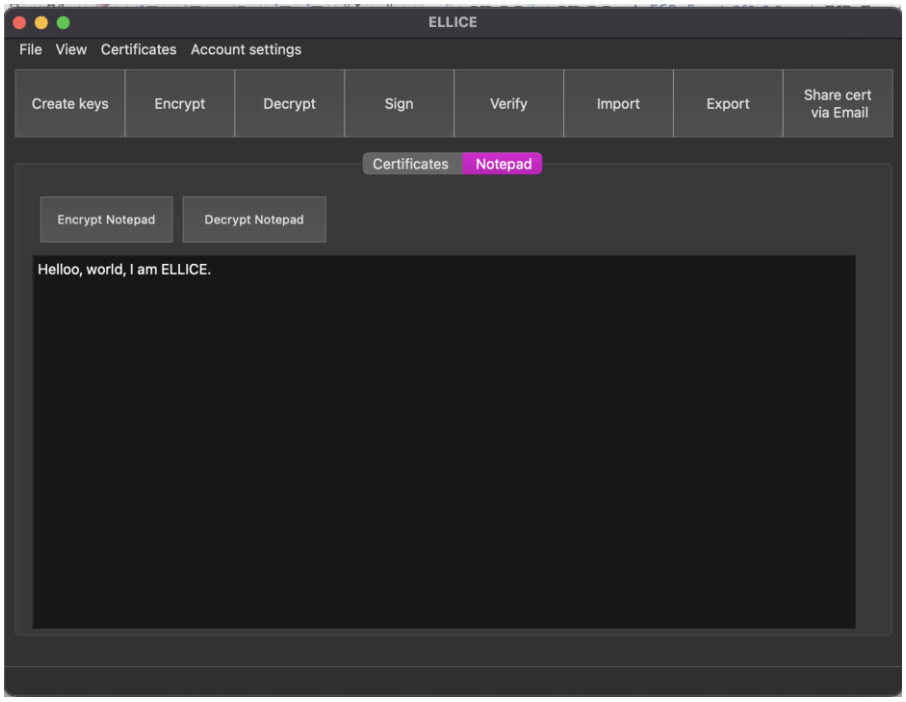

Рисунок 3.20 – Розшифрували повідомлення

3.5 Підпис та верифікація підписаних документів

За допомогою програми ELLICE можна підписувати та верифіковувати електронним підписом документи. Для цього використовується алгоритм ECDSA,

який описаний в розділі 2.4.1. Для того щоб створити електронний цифровий підпис треба (рисунки 3.21-3.22):

1. Створити сертифікат та обрати його;
2. Натиснути кнопку «Sign»;
3. Обрати документ на підпис;
4. Отримуємо повідомлення про успішне підписання (рисунок 3.23).

Рисунок 3.21 – Файл для підпису

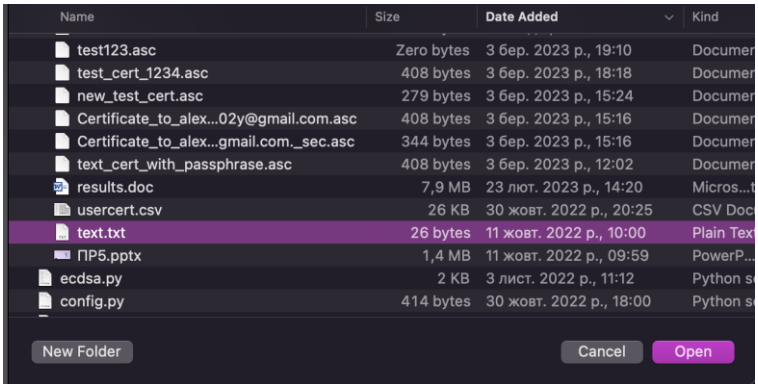

Рисунок 3.22 – Обираємо файл

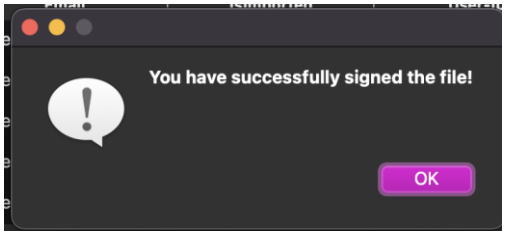

Рисунок 3.23 – Повідомлення про успішне підписання документу

Після підписання створюється файл, за допомогою якого можна перевірити чи був модифікований документ. Даний файл має передаватись разом з документом та перевіряється при отриманні. Файл з сигнатурою зображений на рисунку 3.24.

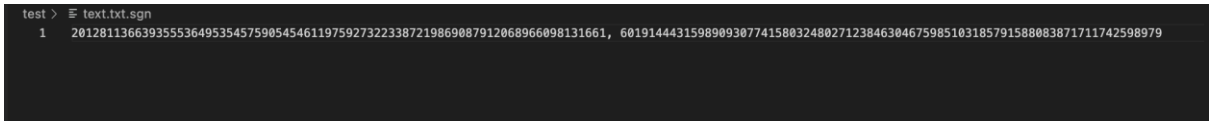

Рисунок 3.24 – Файл з сигнатурою

Для створення підпису використовується алгоритм ECDSA, а саме функція «sign» та функція «hash_message». Після виходу з функції «Sign» отримуємо два значення, що називаються сигнатурою, саме за допомогою сигнатури перевіряється порушення цілісності документу. Сигнатура записується в файл з розширенням «.sgn» та передається разом з документом. При її створенні використовується приватний ключ, а при перевірці необхідний публічний ключ. Тому велика увага має приділятися захисту приватного ключа. Шляхи захисту описані у розділах 3.3, 3.6.

Для того щоб перевірити підписаний документ треба (рисунки 3.25-3.28):

1. Створити сертифікат та обрати його
2. Натиснути кнопку «Verify»
3. Обрати документ, який містить сигнатуру (рисунок 3.25). Важливо, щоб файл який перевіряється, знаходився у поточній директорії та мав таку саму назву, але без «.sgn», по можливості самостійно змінювати назви файлів не треба.
4. Якщо файл не був модифікований, отримуємо повідомлення про успіх та ім’я підписанта. Якщо був модифікований, відповідно підпис буде не буде дійсний.

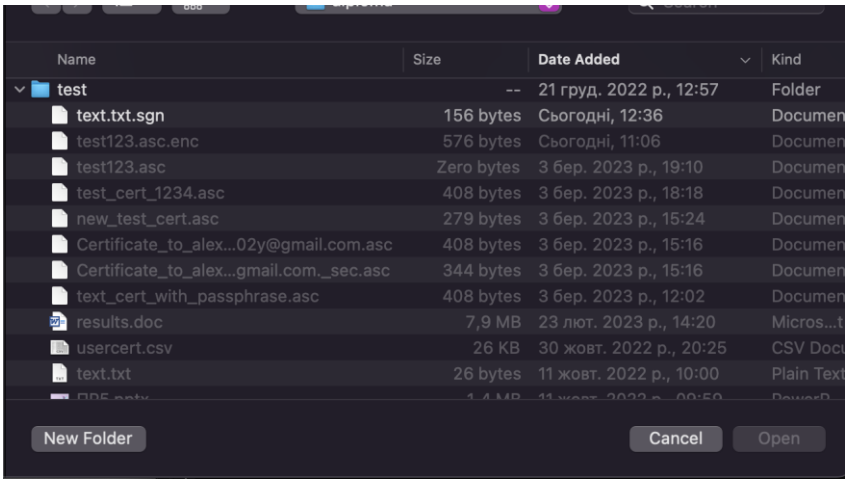

Рисунок 3.25 – Обираємо файл з сигнатурою для перевірки підпису

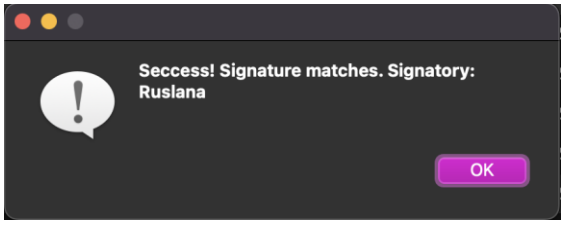

Рисунок 3.26 – Успішна перевірка підпису, файл не був модифікований

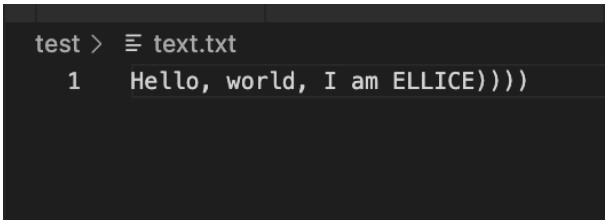

Рисунок 3.27 – Змінюємо вміст файлу

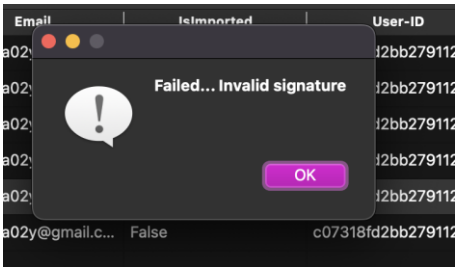

Рисунок 3.28 – Сигнатури не співпадають

Для перевірки підпису виконуються такі дії: зчитуються два значення сигнатури з обраного файлу; зчитаються по-бітово вміст файлу, який був підписаний та формується його геш; на основі сигнатури, гешу та відкритого ключа формується значення, яке має співпадати з першим значенням сигнатури. Якщо вони не співпадають, це означає, що файл був модифікований.

3.6 Обмін сертифікатами

В розробленому програмному забезпеченні ELLICE є можливість експортувати та імпортувати сертифікати. У більшості випадків при використанні програми для шифрування або підпису документів, користувач передає документи відкритим

каналом та для того щоб отримувач мав змогу розшифрувати або перевірити підпис, було створено систему імпортування та експортування створених сертифікатів.

Також важливим моментом є забезпечення безпечної передачі сертифікатів відкритим каналом. Для цього використовуємо захист за допомогою passphrase. Passphrase – це умовний пароль, за допомогою якого шифрується експортований файл з сертифікатом.

Щоб експортувати сертифікат треба обрати сертифікат та натиснути на кнопку «export», після чого з'являється наступне вікно, зображено на рисунку 3.29. У цьому вікні треба ввести passphrase та підтвердити. Це забезпечить безпеку всієї інформації про сертифікат.

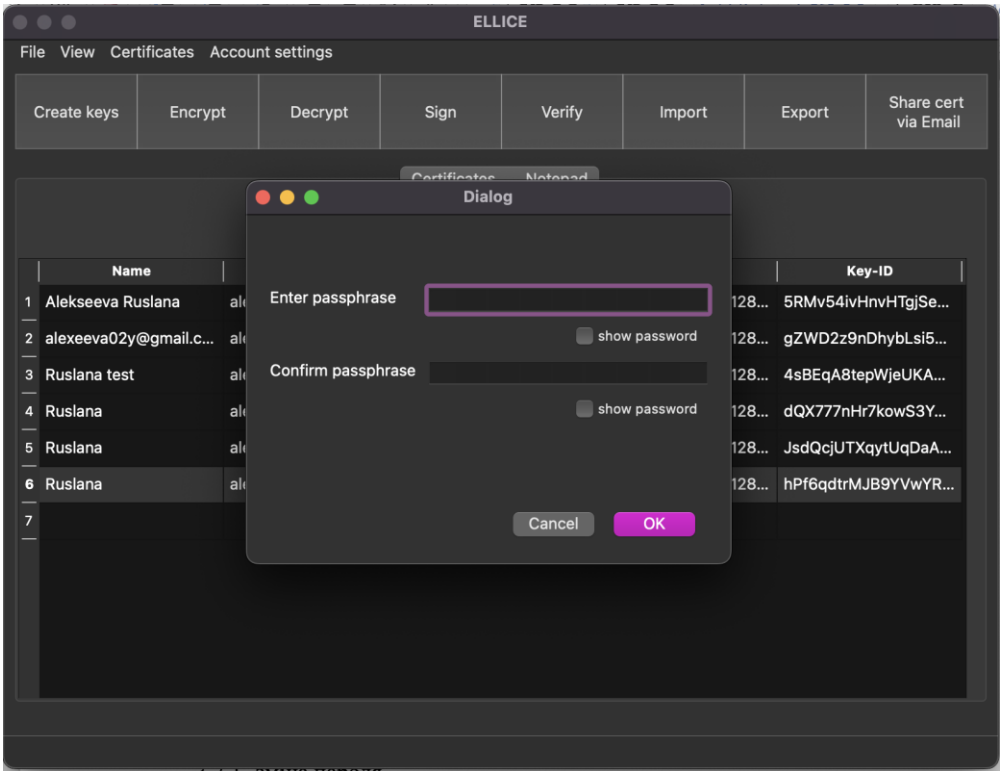

Рисунок 3.29 – Вікно для вводу passphrase

Далі обираємо директорію куди зберегти сертифікат (рисунок 3.30).

Після успішного імпортування отримуємо повідомлення зображено на рисунку 3.31.

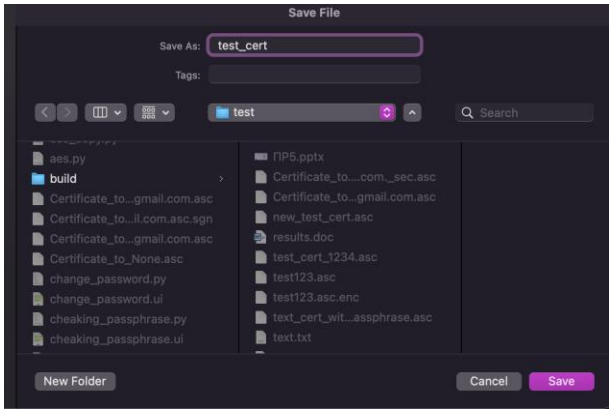

Рисунок 3.30 – Зберігаємо файл

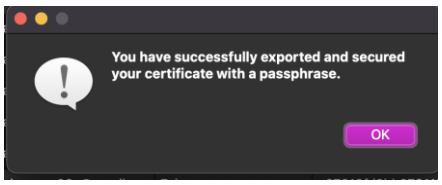

Рисунок 3.31 – Повідомлення про успішне експортування захищеного сертифікату

Такий вигляд має збережений та захищений сертифікат (рисунок 3.32).

Рисунок 3.32 – Зашифрований сертифікат

Для того щоб імпортувати сертифікат треба натиснути на кнопку «Import», обрати сертифікат з директорії (рисунок 3.33).

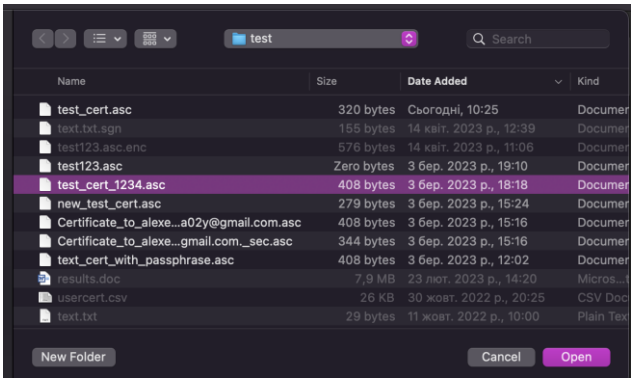

Рисунок 3.33 – Обираємо файл з сертифікатом для імпорту

Далі треба ввести passphrase (рисунок 3.34), якщо пароль правильний сертифікат успішно імпортується та з’явиться в переліку сертифікатів в програмі та зберігається сховищі поточного користувача. Якщо пароль неправильний сертифікат не імпортується.

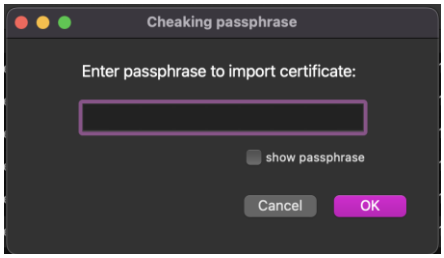

Рисунок 3.34 – Вікно для вводу passphrase

Важливо зазначити, яка інформація імпортується з сертифікату:

- Id користувача, що створив сертифікат
- Ім’я
- Пошта
- Ідентифікатор сертифікату
- Назва еліптичної кривої
- Режим роботи
- Дата створення сертифікату
- Кількість днів доступності сертифікату з дати створення
- Два відкритих ключі

Приватний ключ не імпортується, він належить лише користувач, що створив сертифікат. Для розшифрування зашифрованого файлу використовуються лише відкриті ключі, так само і для перевірки підпису. Якщо зловмисник або будь-який користувач отримає публічний ключ з сертифікату іншого користувача, то він зможе підписати документи від імені іншої людини. Тому імпорт закритого ключа неможливий.

Також є можливість поділитися сертифікатом за допомогою пошти. Щоб це зробити треба обрати сертифікат з переліку в програмі, натиснути на кнопку «Share cert via email», після чого ввести пошту отримувача (рисунок 3.35) та ввести

passphrase для захисту (рисунок 3.36). Після чого отримуємо повідомлення про успішне відправлення (рисунок 3.37).

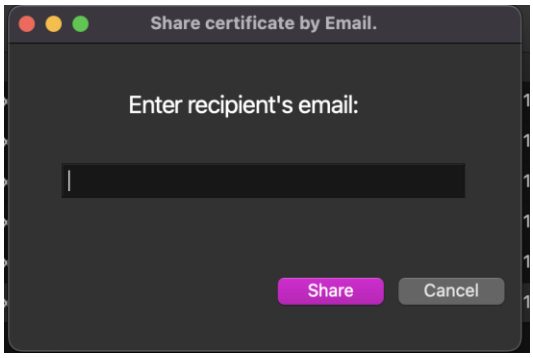

Рисунок 3.35 – Вікно для вводу пошти отримувача

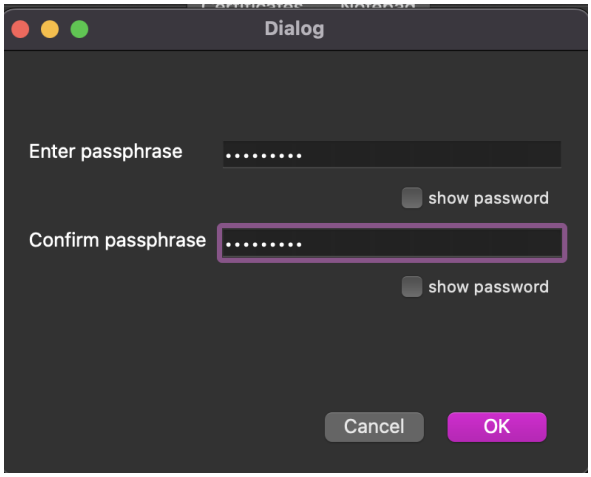

Рисунок 3.36 – Вікно для вводу passphrase

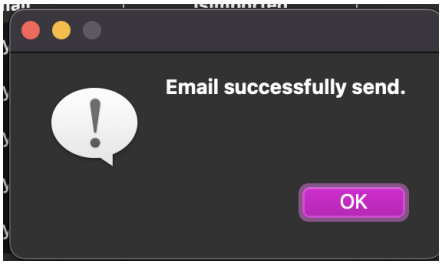

Рисунок 3.37 – Повідомлення про успіх

Перейдемо до пошти та перевіримо лист (рисунок 3.38). Отримали лист з сертифікатом. Листи відправляються від створеного акаунта для програми ELLICE.

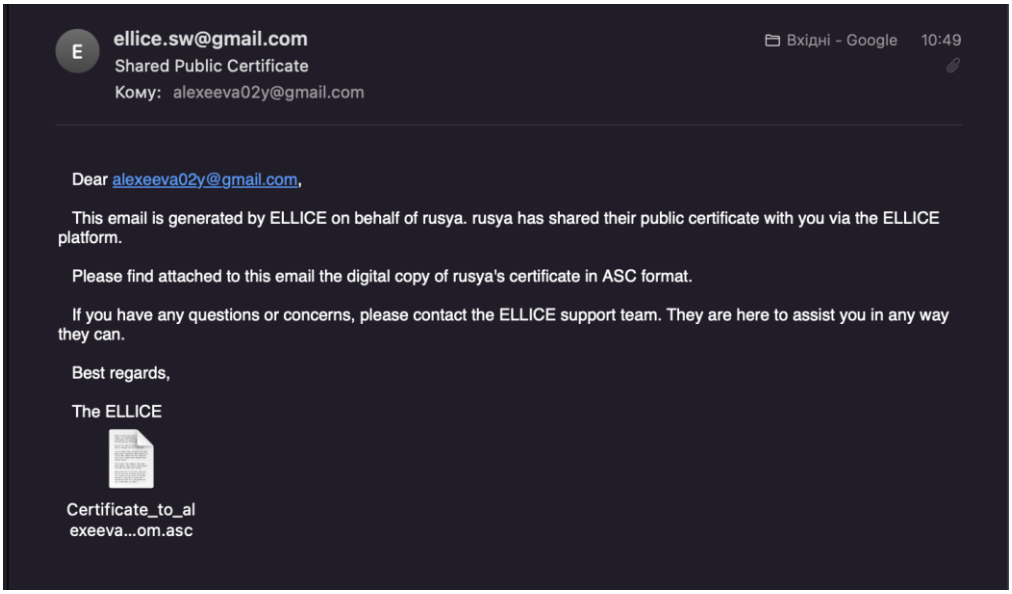

Рисунок 3.38 – Лист з сертифікатом

3.7 Управління акаунтами

Програма ELLICE підтримує можливість використання декількох акаунтів на одному комп’ютері. Для цього були розроблені деякі допоміжні модулі для управління акаунтами, які знаходяться в панелі меню «Account settings».

3.7.1 Зміна пароля

Для того, щоб змінити пароль поточного користувача переходимо в панелі меню «Account settings» та натискаємо «Change password» (рисунок 3.39). Після чого вводимо поточний пароль, новий пароль та підтверджуємо новий пароль. Якщо поточний пароль правильний, то зміна пароля на новий застосується (рисунок 3.40).

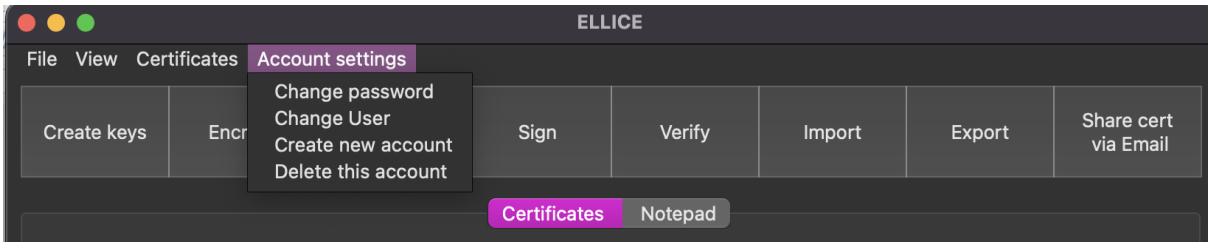

Рисунок 3.39 – Меню «Account settings» -> «Change password»

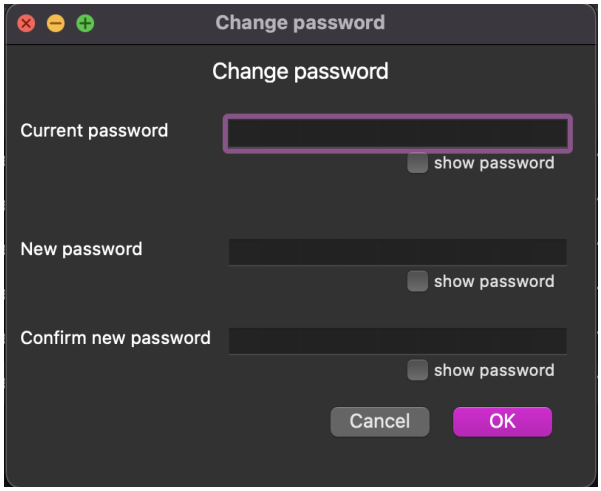

Рисунок 3.40 – Вікно для зміни пароля

Як було зазначено вище паролі користувачів не зберігаються в файлах, за допомогою пароля шифрується вміст сховища сертифікатів користувача. Тож при зміні пароля, програма намагається розшифрувати введеним паролем файл, якщо успішно, зашифровує новим паролем та зберігає. Якщо не успішно, повідомляє про помилку в паролі.

3.7.2 Зміна користувача

У програмі є можливість зміни облікового запису поточного користувача на інший. Для цього необхідно в панелі меню «Account settings» та натискаємо «Change user». Програма переводить користувача у вікно з логіном (рисунок 3.41).

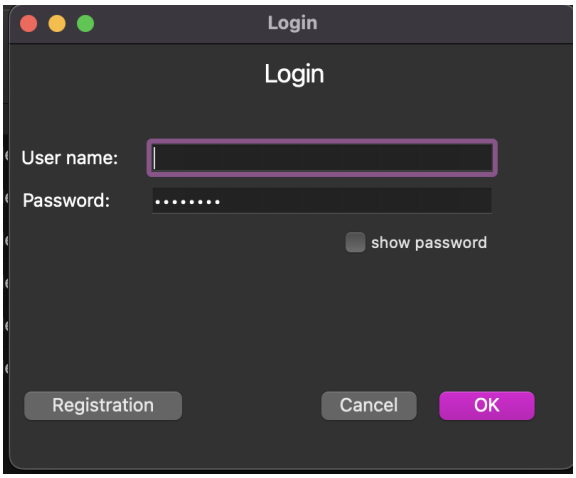

Рисунок 3.41 – Вікно логіну

Після проходження авторизації, обліковий запис змінюється. Якщо авторизація не була успішною, користувач залишається в своєму початковому обліковому записі (рисунок 3.42).

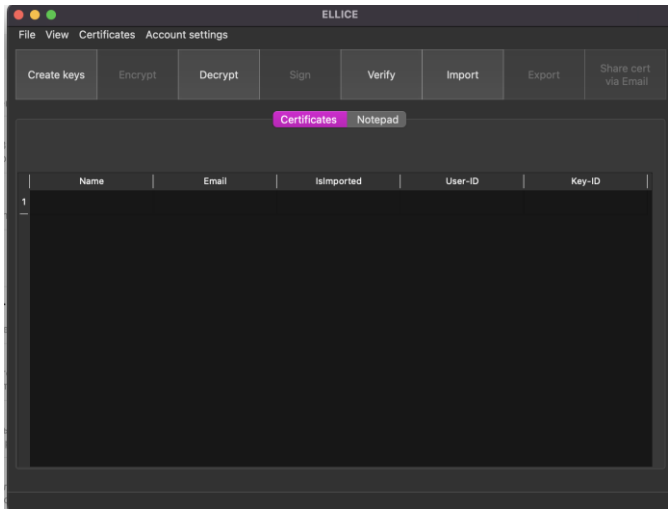

Рисунок 3.42 – Успішна авторизація

3.7.3 Створення нового акаунту

У програмі є можливість створити новий акаунт під час поточного сеансу. Для цього необхідно в панелі меню «Account settings» та натискаємо «Create new account». Програма переводить користувача у вікно з реєстрацією користувача (рисунок 3.43). Після проходження реєстрації аналогічно до реєстрації описаної в розділі 3.1, користувач має пройти авторизацію і тоді отримати доступ до створеного акаунту (рисунки 3.44-3.45).

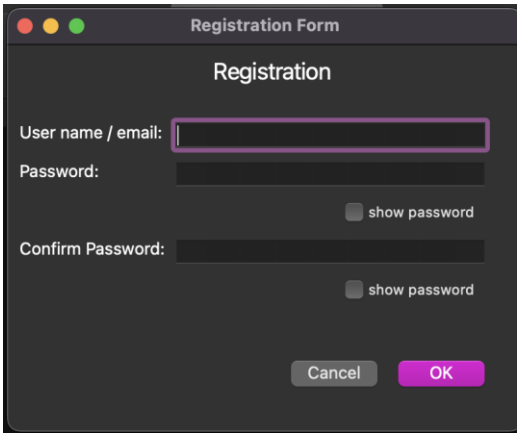

Рисунок 3.43 – Вікно реєстрації

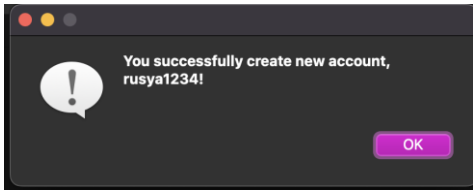

Рисунок 3.44 – Повідомлення про успішне створення нового акаунту

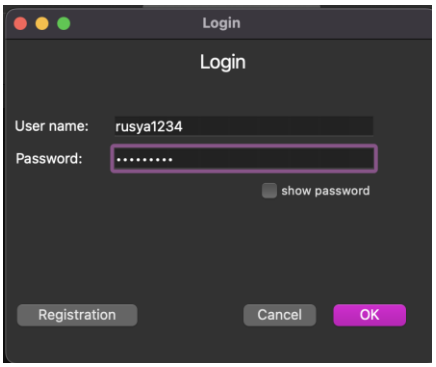

Рисунок 3.45 – Логін в новий акаунт

3.7.4 Видалення поточного акаунту

У програмі є можливість видалити поточний акаунт. Для цього необхідно в панелі меню «Account settings» та натискаємо «Delete this account». Файл який створюється при реєстрації та який оновлюється під час створення нових сертифікатів, видаляється та з’являється повідомлення про успішне видалення акаунту (рисунок 3.46).

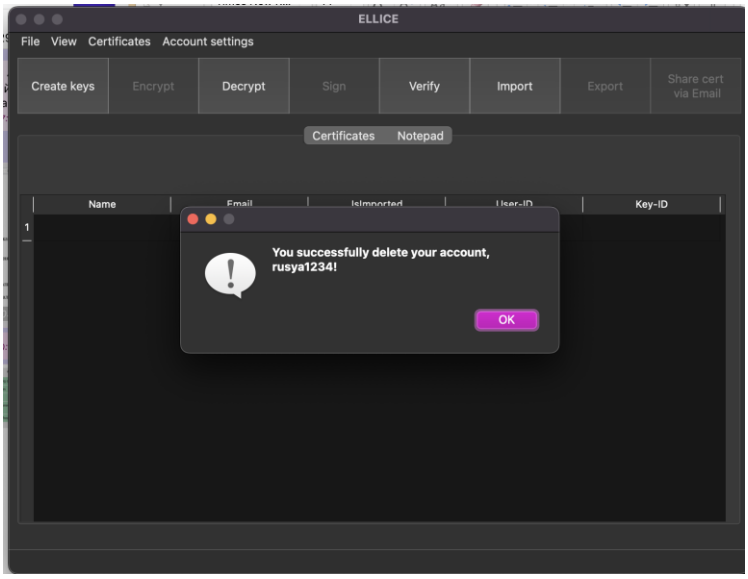

Рисунок 3.46 – Повідомлення про видалення поточного акаунту

Після чого головне вікно програми закривається і з’являється меню для авторизації (як при початковому запуску програми) (рисунок 3.47).

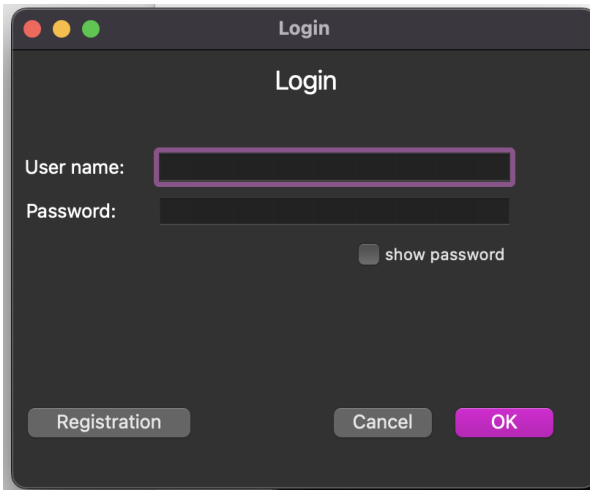

Рисунок 3.47 – Вікно логіну

3.8 Висновки

У даному розділі були розглянуті основні програмні особливості створеного застосунку, відповідно до вимог описаних у розділі 2.1, для забезпечення цілісності та конфіденційності даних, шляхом використання алгоритму шифрування AES у поєднанні з еліптичною криптографією та алгоритму ECDSA для створення та перевірки цифрового підпису. У даному розділі були описані та протестовані функціональні модулі, такі як авторизація та реєстрація користувачів, система створення та зберігання сертифікатів, шифрування та розшифрування файлів та тексту, підпис та перевірка підпису, види обміну сертифікатів такі як імпорт, експорт та пересилка сертифікатів поштою, також розкриті особливості управління акаунтами у програмному забезпеченні ELLICE.

ВИСНОВКИ

У кваліфікаційній роботі було реалізовано програмний засів для забезпечення цілісності та конфіденційності даних за рахунок використання таких криптографічних механізмів, як алгоритм шифрування AES, алгоритм генерація ключів на основі еліптичних кривих та алгоритм створення та перевірки цифрового підпису ECDSA.

У першій частині кваліфікаційної роботи було проведено аналіз необхідність забезпечення безпечного обміну інформації шляхом криптографічних механізмів для забезпечення цілісності та конфіденційності даних. Проведено порівняння блокових алгоритмів шифрування, в результаті чого алгоритм AES був обраний як найкращий з урахуванням рівня безпеки, продуктивності та універсальності. Для забезпечення більшої криптостійкості ключів AES було обрано використання алгоритму генерації ключів на основі еліптичних кривих. Також було розглянуто використання алгоритму ECDSA для створення та перевірки електронних цифрових підписів. Описані алгоритми та їх особливості використовуються в подальшому дослідженні та розробці програмного забезпечення з метою забезпечення безпеки та захисту.

У другій частині кваліфікаційної роботи було розглянуто вимоги до програмного забезпечення, його функціональні модулі та взаємодію між ними. Також були досліджені програмні реалізації криптографічних алгоритмів, аналізовані можливості комбінування криптографічних методів для забезпечення необхідного рівня безпеки. З метою підвищення криптостійкості було обрано поєднання алгоритму шифрування AES з алгоритмом генерації ключів на основі еліптичних кривих (ECC).

У третій частині кваліфікаційної роботи на основі проаналізованих у першій та другій частині даних було розроблено програмний засіб відповідно до поставлених вимог, були розглянуті основні програмні особливості створеного застосунку, такі як авторизація та реєстрація користувачів, система створення та зберігання сертифікатів, шифрування та розшифрування файлів та тексту, підпис та перевірка підпису, види

обміну сертифікатів такі як імпорт, експорт та пересилка сертифікатів поштою, також розкриті особливості управління акаунтами у програмному забезпеченні ELLICE.

Виходячи із поставленої мети кваліфікаційної роботи були виконані наступні завдання:

- Розглянуті основні принципи шифрування та електронного цифрового підпису.
- Досліджено теоретичні аспекти блочних алгоритмів шифрування та еліптичних кривих.
- Розроблений програмний засіб для забезпечення цілісності та конфіденційності даних.
- Реалізований блочний алгоритм шифрування та електронний цифровий підпис на основі еліптичних кривих у програмному засобі.
- Проведено тестування розробленого програмного засобу.

Всі задачі було виконано в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rajdeep B. A Review and Comparative Analysis of Various Encryption Algorithms / B. Rajdeep, H. Rahul. // International Journal of Security and Its Applications. – 2015. – С. 289–306.

2. The DES encryption algorithm [Електронний ресурс] // Ius mentis. – 2003. – Режим доступу до ресурсу: https://www.iusmentis.com/technology/encryption/des/.

3. Sharma M. Comparative Analysis of Block Key Encryption Algorithms / M. Sharma, R. Garg. // International Journal of Computer Applications. – 2016. – С. 26–35.

4. International Data Encryption Algorithm (IDEA) [Електронний ресурс] – Режим доступу до ресурсу: https://opencourses.emu.edu.tr/pluginfile.php/47495/mod_resource/content/2/Block%20ciphers%20(IDEA).pdf.

5. Blowfish (cipher) [Електронний ресурс] // 2007 – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Blowfish_(cipher)#cite_note-schneier-interview-dec-2007-3.

6. Advanced Encryption Standard (AES), 2001. – 38 с. – (Federal Information Processing Standards Publication). – (FIPS 197).

7. Kak A. AES: The Advanced Encryption Standard [Електронний ресурс] / Avi Kak // Purdue University. – 2023. – Режим доступу до ресурсу: https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf.

8. Wang S. The difference in five modes in the AES encryption algorithm [Електронний ресурс] / Shawn Wang. – 2019. – Режим доступу до ресурсу: https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/.

9. Технології захисту інформації [Електронний ресурс] // Харківський національний економічний університет – Режим доступу до ресурсу: https://studfile.net/preview/5993348/page:13/.

10. The Case for Elliptic Curve Cryptography [Електронний ресурс] // National Security Agency. – 2009. – Режим доступу до ресурсу:

https://web.archive.org/web/20090117023500/http:/www.nsa.gov/business/programs/elliptic_curve.shtml.

11. Elliptic-curve cryptography [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Elliptic-curve_cryptography.

12. Elliptic curve point multiplication [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication.

13. Lynn B. Explicit Addition Formulae [Електронний ресурс] / Ben Lynn – Режим доступу до ресурсу: https://crypto.stanford.edu/pbc/notes/elliptic/explicit.html.

14. Elliptic Curve Cryptography., 2009. – 97 с. – (SEC).

15. Ilyenko A. Перспективи використання еліптичної криптографії для забезпечення цілісності та конфіденційності інформації / A. Ilyenko, S. Ilyenko, Y. Masur. // Вісник університету "Україна". – №23.

16. Закон України Про електронні довірчі послуги [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2155-19#Text.

17. Blockchain – Elliptic Curve Digital Signature Algorithm (ECDSA) [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/blockchain-elliptic-curve-digital-signature-algorithm-ecdsa/.

18. Криптографічний система RSA [Електронний ресурс] – Режим доступу до ресурсу: https://stud.com.ua/179775/informatika/kriptografichniy_sistema.

19. RSA [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/RSA.

20. ECDSA vs RSA: Everything You Need to Know [Електронний ресурс] // InfoSec Insights. – 2020. – Режим доступу до ресурсу: https://sectigostore.com/blog/ecdsa-vs-rsa-everything-you-need-to-know/.

21. Gills A. Digital signature [Електронний ресурс] / A. Gills, B. Lutkevich, V. Brunskill // TechTarget – Режим доступу до ресурсу: https://www.techtarget.com/searchsecurity/definition/digital-signature.

22. Digital Signature Standard (DSS), 2023. – 77 с. – (FIPS 186-5). – (186-5).

23. Johnson D. The Elliptic Curve Digital Signature Algorithm (ECDSA) [Електронний ресурс] / D. Johnson, A. Menezes. – 1999. – Режим доступу до ресурсу: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c26e3c42c2a85e2479c1316ccc8c20754533e406.

24. Dimitrov V. Fast elliptic curve point multiplication using double-base chains. / V. Dimitrov, L. Imbert, P. Mishra. – 2005.

25. SafeCurves: choosing safe curves for elliptic-curve cryptography [Електронний ресурс] – Режим доступу до ресурсу: https://safecurves.cr.yp.to.

26. Command Line Elliptic Curve Operations [Електронний ресурс] // OpenSSL – Режим доступу до ресурсу: https://wiki.openssl.org/index.php/Command_Line_Elliptic_Curve_Operations.

27. Elliptic Curve Cryptography: a gentle introduction [Електронний ресурс] – Режим доступу до ресурсу: https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/.

28. A Byte of Python [Електронний ресурс] – Режим доступу до ресурсу: https://python.swaroopch.com.

29. Willman J. Beginning PyQt [Електронний ресурс] / J. M. Willman. – 2022. – Режим доступу до ресурсу: https://doi.org/10.1007/978-1-4842-7999-1.

ДОДАТОК 1

Програмний код генерації ключів на основі еліптичних кривих

```
import collections
import random
from egcd import egcd

EllipticCurve = collections.namedtuple('EllipticCurve', 'name p a b g n h')

DEFAULT_ELLIPTIC_CURVE = EllipticCurve(
    'secp256k1',
    p=0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f,
    a=0,
    b=7,
    g=(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,
       0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8),
    n=0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141,
    h=1,
)

ELLIPTIC_CURVE_160k1 = EllipticCurve(
    'secp160k1',
    p=0xffffffffffffffffffffffffffffffffefffac73,
    a=0,
    b=7,
    g=(0x3b4c382ce37aa192a4019e763036f4f5dd4d7ebb,
       0x938cf935318fdced6bc28286531733c3f03c4fee),
    n=0x100000000000000000001b8fa16dfab9aca16b6b3,
    h=1,
)


ELLIPTIC_CURVE_192k1 = EllipticCurve(
    'secp192k1',
    p=0xfffffffffffffffffffffffffffffffffffffffeffffee37,
    a=0,
    b=3,
    g=(0xdb4ff10ec057e9ae26b07d0280b7f4341da5d1b1eae06c7d,
       0x9b2f2f6d9c5628a7844163d015be86344082aa88d95e2f9d),
    n=0xfffffffffffffffffffffffe26f2fc170f69466a74defd8d,
    h=1,
)


ELLIPTIC_CURVE_224k1 = EllipticCurve(
    'secp224k1',
    p=0xfffffffffffffffffffffffffffffffffffffffffffffffeffffe56d,
    a=0,
    b=5,
    g=(0xa1455b334df099df30fc28a169a467e9e47075a90f7e650eb6b7a45c,
       0x7e089fed7fba344282cafbd6f7e319f7c0b0bd59e2ca4bdb556d61a5),
    n=0x010000000000000000000000000001dce8d2ec6184caf0a971769fb1f7,
    h=1,
```

```
)

curves = {"secp256k1": DEFAULT_ELLIPTIC_CURVE,
"secp160k1":ELLIPTIC_CURVE_160k1, 'secp192k1':ELLIPTIC_CURVE_192k1,
'secp224k1':ELLIPTIC_CURVE_224k1}

class ECCKeyGenerator:
    def __init__(self, private_key=None, curve: EllipticCurve =
DEFAULT_ELLIPTIC_CURVE) -> None:
        if private_key is None:
            private_key = random.randrange(1, curve.n)
        self.private_key = private_key
        self.curve = curve

    def gen_keypair(self):
        return self.private_key, self.scalar_mult(self.private_key,
self.curve.g)

    @staticmethod
    def extended_gcd(k, p):
        if k == 0:
            raise ZeroDivisionError('division by zero')
        if k < 0:
            return p - ECCKeyGenerator.extended_gcd(-k, p)
        gcd, x, _ = egcd(k, p)
        assert gcd == 1
        assert (k * x) % p == 1
        return x % p


    def is_on_curve(self, point):
        if point is None:
            return True
        x, y = point
        return (y * y - x * x * x - self.curve.a * x - self.curve.b) %
self.curve.p == 0


    def point_neg(self, point):
        assert self.is_on_curve(point)
        if point is None:
            return None
        x, y = point
        result = (x, -y % self.curve.p)
        assert self.is_on_curve(result)
        return result


    def add_point(self, point1, point2):
        assert self.is_on_curve(point1)
        assert self.is_on_curve(point2)

        if point1 is None:
            return point2
        if point2 is None:
            return point1

        x1, y1 = point1
```

```
        x2, y2 = point2

        if x1 == x2 and y1 != y2:
            return None
        if x1 != x2:
            lmd = (y1 - y2) * ECCKeyGenerator.extended_gcd(x1 - x2,
self.curve.p)
        else:
            lmd = (3 * x1**2 + self.curve.a) * ECCKeyGenerator.extended_gcd(2
* y1, self.curve.p)

        x3 = lmd * lmd - x1 - x2
        y3 = y1 + lmd * (x3 - x1)
        result = (x3 % self.curve.p, -y3 % self.curve.p)
        assert self.is_on_curve(result)
        return result


    def scalar_mult(self, k, point):
        assert self.is_on_curve(point)
        if k % self.curve.n == 0 or point is None:
            return None
        if k < 0:
            return self.scalar_mult(-k, self.point_neg(point))

        result = None
        addend = point
        while k:
            if k & 1:
                result = self.add_point(result, addend)
            addend = self.add_point(addend, addend)
            k >>= 1
        assert self.is_on_curve(result)
        return result



if __name__ == "__main__":
    ecc = ECCKeyGenerator(curve=DEFAULT_ELLIPTIC_CURVE)
    print('Curve:', ecc.curve.name)
    private_key, public_key = ecc.gen_keypair()
    print(hex(private_key))
    print(hex(public_key[0]))
    print(hex(public_key[1]))
```