МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

НАСІКОВСЬКИЙ ВІТАЛІЙ ВАЛЕРІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____ О. В. Зелінська
«_____» _____ 2024 р.

РОЗРОБКА ДОДАТКУ ДЛЯ ЗБЕРІГАННЯ ТА КЕРУВАННЯ ЗАШИФРОВАНИМИ ФАЙЛАМИ

Спеціальність 122 «Комп’ютерні науки»
Кваліфікаційна (бакалаврська) робота

Керівник:
Зелінська О. В., доцент кафедри
інформаційних технологій,
к.т.н., доцент

Оцінка: ___/___/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2024

АНОТАЦІЯ

Насіковський В. В. Розробка додатку для зберігання та керування зашифрованими файлами. Спеціальність 122 «Комп’ютерні науки» Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній (бакалаврській) роботі досліджено можливості та актуальність застосування специфікації шифрування AES для забезпечення безпеки даних у рамках розробки додатку на мові програмування Python.

Описано процес проектування та реалізації додатку, який дозволяє шифрувати текст та файли, використовуючи алгоритм, побудований на основі бібліотеки 'cryptography', та розглянуто основні аспекти побудови графічного інтерфейсу користувача за допомогою бібліотеки 'PySide6'.

Ключові слова: AES, шифрування, Python, 'cryptography', 'PySide6'.

50 с., 28 рис., 27 джерел.

ABSTRACT

Nasikovskyi V. V. Development of an application for storing and managing encrypted files. Specialty 122 "Computer Sciences" Vasyl’ Stus Donetsk National University, Vinnytsia, 2024.

This bachelor's thesis explored the potential and relevance of using the AES encryption specification to ensure data security within the development of an application using the Python programming language.

The application's design and implementation process, which allows for the encryption of text and files using an algorithm built on the cryptography library, is described. Furthermore, the main aspects of creating a graphical user interface with the PySide6 library are examined.

Keywords: AES, encryption, Python, 'cryptography', 'PySide6'.

50 p., 28 figs., 27 refs.

ЗМІСТ

ВСТУП

У сучасному світі, де величезний обсяг інформації щодня передається через глобальні мережі, питання її захисту стає все більш актуальним і значущим. Від найдавніших часів людина прагнула захистити свої знання та дані, розуміючи, що володіння інформацією означає володіння ресурсом неоціненної ваги. "Хто володіє інформацією, той володіє світом" — цей відомий вислів набуває нових значень в контексті сучасної цифрової ери, де не захищена інформація стає легкодоступною для несанкціонованого використання [1].

Програмне шифрування в даному випадку стає фундаментом безпеки інформації, необхідним для забезпечення її конфіденційності та цілісності. Ця техніка перетворює відкриті дані в зашифрований текст, який можна розшифрувати лише за допомогою спеціального ключа, доступного тільки уповноваженим особам. Відповідно, в контексті постійно зростаючих загроз у сфері кібербезпеки, застосування алгоритмів шифрування разом з розширеними криптографічними ключами виступають як важлива складова кожного, хто прагне досягти надійного збереження інформації.

Актуальність проблеми захисту інформації з кожним днем лише зростає через стрімкий розвиток технологій та появу квантових комп'ютерів, які здатні розшифровувати традиційні алгоритми шифрування набагато швидше, ніж це було можливо раніше. Ця нова реальність ставить під сумнів абсолютні гарантії безпеки, що раніше вважалися непохитними. У таких умовах, постійне оновлення та використання останніх досягнень у галузі алгоритмів шифрування стає ключовим елементом у стратегіях захисту.

Мета бакалаврської роботи: розробка надійного інструменту з відкритим кодом, який інтегрує останні та найактуальніші алгоритми захисту. Цей інструмент буде оснащений інтуїтивно зрозумілим інтерфейсом для користувача, забезпечуючи можливість швидкого та ефективного шифрування та дешифрування файлів та тексту.

Завдання дослідження:

- Розгляд методів захисту даних для протидії кібератакам.
- Дослідження процесу шифрування, зосереджуючись на принципі роботи алгоритму AES.
- Огляд існуючих аналогів та визначення їх основних переваг та недоліків.
- Аналіз доступних інструментів та бібліотек, для реалізації програмного продукту;
- Розробка програмного забезпечення та його тестування;

Об'єкт дослідження: програмне забезпечення, яке реалізує функції шифрування та дешифрування даних з використанням алгоритму AES.

Предмет дослідження: принципи роботи, ефективність та застосування сучасних алгоритмів шифрування, у контексті розробки програмного забезпечення для криптографічного захисту інформації.

Практичне значення: програма буде використовуватись для забезпечення високого рівня безпеки особистих та корпоративних даних, шляхом їх шифрування та дешифрування. Це дозволить користувачам безпечно зберігати важливу інформацію, обмінюватися нею через мережу або передавати через ненадійні канали, знижуючи ризик несанкціонованого доступу та витоку даних.

Структура роботи: У вступі детально розглядається актуальність теми захисту інформації у сучасному цифровому світі. У першому розділі зосереджено увагу на теоретичних аспектах шифрування, детально розглядається, що таке AES, як працює цей алгоритм, його ключові особливості та переваги перед іншими алгоритмами. Другий розділ присвячений практичній реалізації проекту, включаючи постановку задачі, вибір технологій та опис використаних бібліотек. У третьому розділі розглядається практичне застосування розробленої програми, з наданням прикладів шифрування. У висновках описані основні результати роботи.

РОЗДІЛ 1
ОГЛЯД СУЧАСНИХ МЕТОДІВ ШИФРУВАННЯ ДАНИХ ТА ЇХ АНАЛІЗ

1.1 Поняття шифрування та його значення в контексті безпеки даних

З кожним днем технології розвиваються неймовірно швидко, відкриваючи перед нами нові горизонти можливостей. Цей стрімкий розвиток супроводжується збільшенням обсягу інформації, що передається між користувачами. У такій динаміці, кіберзлочинці також не залишаються осторонь, використовуючи кожну нагоду, щоб нажитися на недосвідчених користувачах. Вони легко можуть перетворити паролі, дані банківських карт, сімейні фотографії, офіційні документи та інші цінні файли на потужний інструмент для маніпуляцій та скоєння злочинів, якщо ті потраплять у невідповідні руки.

Передусім, критично важливо забезпечити захист інформації, яку ми передаємо через Інтернет. Не менш важливим є і захист від потенційних загроз, які можуть виникнути не тільки онлайн, а й у фізичному світі [2]. Розгляньмо два приклади: флешка, випадково знайдена на вулиці та підключена до нашого пристрою, або електронний лист із невідомим вкладенням або посиланням. Обидва випадки можуть призвести до неконтрольованого витоку даних.

У першій половині 2023 року компанія Mandiant Managed Defense відзначила різке збільшення кількості кібератак через USB-накопичувачі, що утричі перевищило попередні показники. Атаки мали на меті викрадення даних і торкнулися різних секторів по всьому світу. Особливо вирізняються дві кампанії: SOGU, що спрямована на різноманітні галузі в Європі, Азії та США та пов’язана з китайськими шпигунськими діями, і SNOWYDRIVE, націлена на нафтогазову промисловість Азії [3].

Крім збільшення кількості кібератак, компанії стикаються з величезними фінансовими втратами внаслідок витоку даних. Як показує статистика, середня вартість витоку даних за допомогою програм-вимагачів у 2023 році склала 5,13 мільйона доларів, що на 13 відсотків вище, ніж роком раніше, і значно перевищує

середній показник збитків від загального значення, яке становить 4,45 мільйона доларів. За даними IBM, найбільш вразливою сферою виявилась охорона здоров'я, де витрати зросли на 53,3 відсотка і склали 10,93 мільйона доларів [4].

Цікавим є факт, що навіть найсучасніші антивіруси інколи не можуть ефективно протистояти новітнім атакам. Це пов'язано з тим, що хакерські групи постійно розробляють нові скрипти, які антивірусні програми ще не встигають ідентифікувати та додати до своїх баз даних сигнатур [5]. Така ситуація робить актуальною проблему захисту інформації, особливо на тлі стрімкого розвитку штучного інтелекту та квантових технологій.

Постає логічне запитання: як захистити себе від кіберзагроз? Очевидний, але малореалістичний варіант – це уникнення підключення до Інтернету та відмова від використання незнайомих пристроїв. Однак такий підхід не відповідає потребам сучасної людини.

У відповідь на сучасні виклики, людство розробило комплекс заходів, метою яких є мінімізація ризиків витоку інформації. Одним із ключових елементів цієї системи безпеки є шифрування даних. Воно перетворює інформацію у формат, який неможливо прочитати без спеціального ключа, ефективно захищаючи її від неправомірного доступу [6]. Збереження на диску із шифруванням або збереження на хмарі із шифруванням, надійний контроль доступу до пристрою, регулярне резервне копіювання даних та обов'язкове використання останніх версій будь-якого програмного забезпечення. В даний список також входять оновлення від Windows, які всі недолюблюють через неочікувані перезавантаження комп'ютера у зв'язку із встановленням нового патчу.

Наприклад, вразливість CVE-2017-11882 у Equation Editor дозволяє зловмисникам виконувати віддалений код на вразливих пристроях після відкриття шкідливого документа, зазвичай відправленого через фішинговий електронний лист. Ця вразливість, хоч і була розкрита ще у 2017 році, продовжує використовуватись хакерами для доступу до мереж жертв [7]. Відмова від

оновлення системи може спричинити не лише порушення функціонування системи, але й серйозний витік конфіденційних даних.

1.2 Актуальні методи шифрування

Настав час розглянути методи шифрування, які були відомі задовго до появи комп’ютерів. Однією з найдавніших технік шифрування, яка датується ще 1900 роком до нашої ери, є метод заміни символів, виявлений у гробниці Хнумхотепа II в Єгипті. Для розшифровки такого повідомлення потрібен спеціальний ключ. Ще один відомий приклад давніх методів шифрування - це шифр Цезаря, де літера тексту зсувається на певну кількість позицій униз по алфавіту, формуючи закодоване повідомлення [8].

Ці алгоритми, хоч і працюють, але вже не вважаються надійними сьогодні, адже сучасні комп’ютери можуть їх легко розкодувати. З цієї причини перейдемо до сучасних алгоритмів та розглянемо системи симетричного та асиметричного шифрування.

Симетричне шифрування передбачає використання одного ключа всіма сторонами. Це спрощує процес, наприклад, коли ви шифруєте електронний лист унікальним ключем та надсилаєте його іншій людині, вона може використати той самий ключ для розшифровки. Загалом це можна описати, як спільний секрет між двома або більше людьми для підтримки конфіденційного зв'язку [9].

Симетричне шифрування є широко поширеним методом для захисту даних у різноманітних додатках, забезпечуючи конфіденційність та цілісність. Використання єдиного ключа демонструє швидкість та ефективність, на відміну від кількох. Наприклад, застосунки для безпечного обміну повідомлень, такі як Telegram або WhatsApp, використовують симетричне шифрування для забезпечення End-to-End шифрування повідомлень [10-11].

З іншого боку, асиметричне шифрування створено як вдосконалення симетричного методу, вирішуючи ключову проблему безпечної передачі ключа. Ця система використовує два ключі: публічний для шифрування повідомлень та

приватний для їх розшифровки. Особливістю асиметричного шифрування є те, що публічний ключ можна вільно розповсюджувати без ризику для безпеки, адже навіть у разі його перехоплення, без приватного ключа доступ до зашифрованої інформації отримати неможливо. Прикладом може слугувати протоколи SSL/TLS для з'єднань між веб-браузерами та веб-серверами та SSH для безпечної віддаленої роботи з операційними системами через незахищені мережі [12].

Для Python додатку, який буде шифрувати текст та файли, буде використаний AES – симетричний алгоритм шифрування, так як він швидко працює та є одним з найпоширеніших, який застосовується сьогодні. AES був розроблений як заміна застарілому DES (Data Encryption Standard), який було зламано дослідниками безпеки ще у 2005 році [13]. Новий алгоритм було спрямовано на вирішення головної слабкості свого попередника – короткої довжини ключа шифрування, що була вразлива до атак методом brute-force.

1.3 Принцип роботи AES і його ключові характеристики

AES (Advanced Encryption Standard) є симетричним алгоритмом блокового шифрування, який шифрує дані блоками по 128 біт, використовуючи ключі довжиною 128, 192 або 256 біт [13].

AES отримує на вході 128 біт даних і видає на виході 128 біт зашифрованого тексту. Ця технологія використовує принцип, де відбувається послідовна заміна та перемішування вхідних даних.

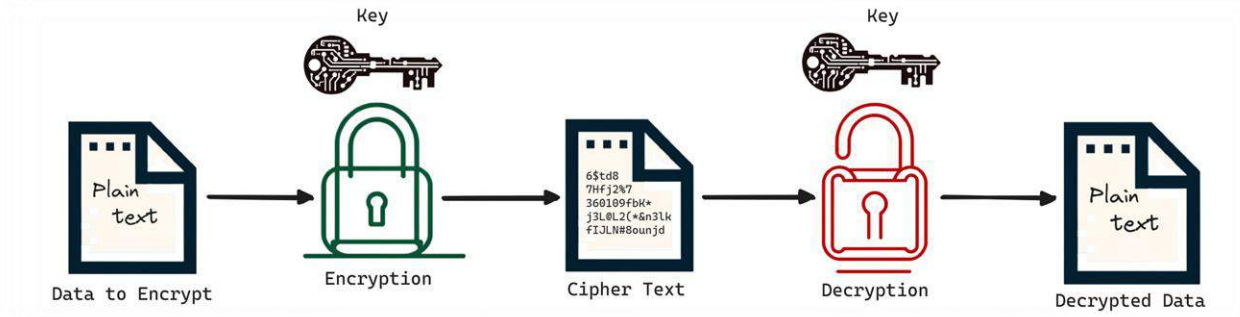


Рисунок 1.1 – Приклад шифрування із однаковим ключем

1.3.1 Огляд основних етапів роботи AES

Розширення ключа (розкладка ключів):

Для початку розберемо, що таке "раунд". В даному випадку "раунд" — це послідовність операцій, що включає в себе різні перестановки, змішування та додавання ключів (рис 1.2). Відповідно, AES починається з процесу розширення ключа, де оригінальний ключ шифрування перетворюється на кілька окремих раундових ключів. Цей процес гарантує використання унікального ключа для кожного раунду шифрування, підвищуючи безпеку. Розширення ключа включає взяття оригінального ключа шифрування та застосування різних перетворень, включаючи заміну S-боксу (substitution box), зсуви та додавання раундової константи.

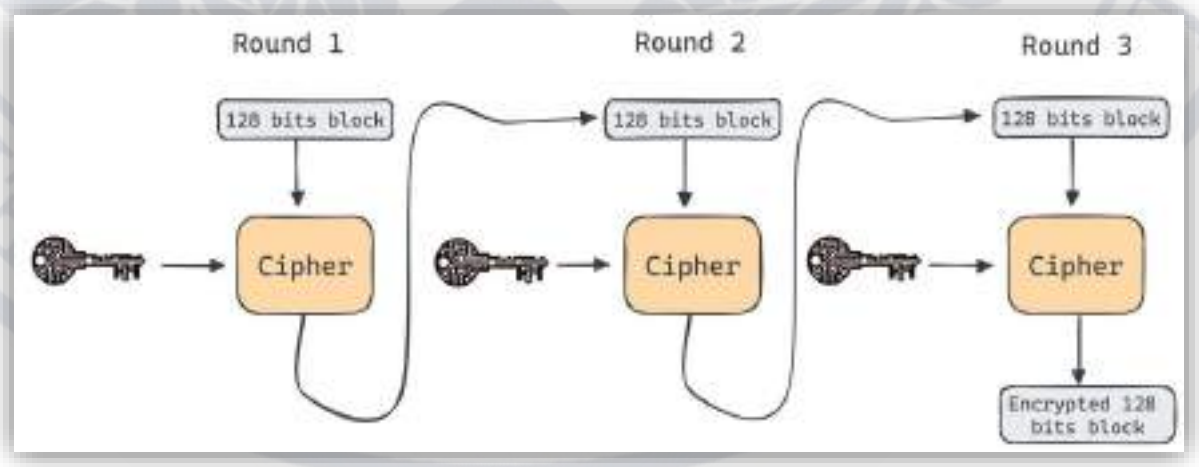


Рисунок 1.2 – Приклад кількох раундів шифрування блоку 128 bit

Початковий раунд:

Початкові дані форматуються у матрицю 4x4, відому як “state matrix” (рис 1.3). Ця матриця піддається операції XOR (виключне АБО), де використовується перший розширений ключ. XOR - це бінарна операція, де біти порівнюються; якщо біти різні, результат буде 1, а якщо однакові - 0. Наприклад, якщо біт у відкритих даних дорівнює 1, а відповідний біт у ключі буде 0, результатом буде 1 (рис 1.4).

Рисунок 1.3 – Операція перетворення “state matrix” та 128 bit ключа у шифр

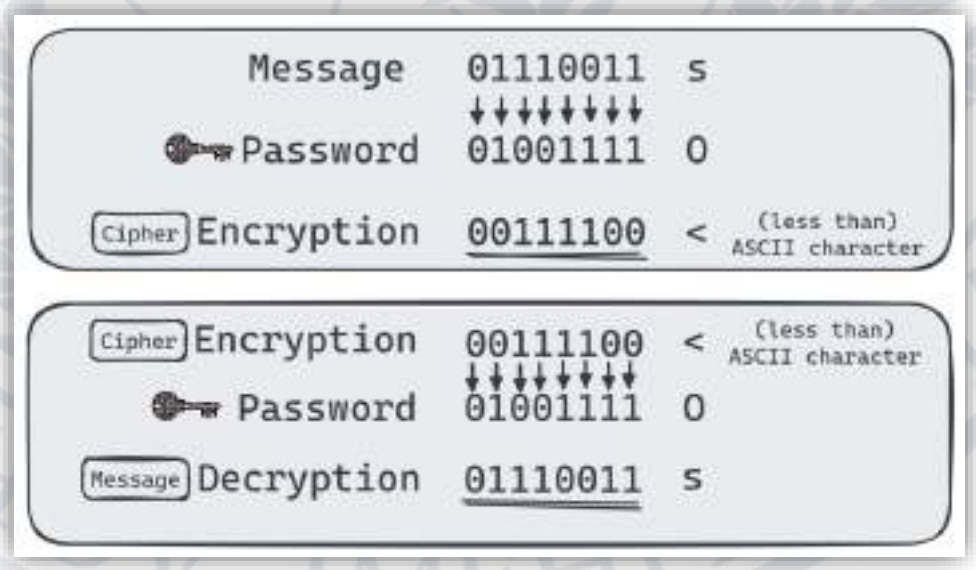
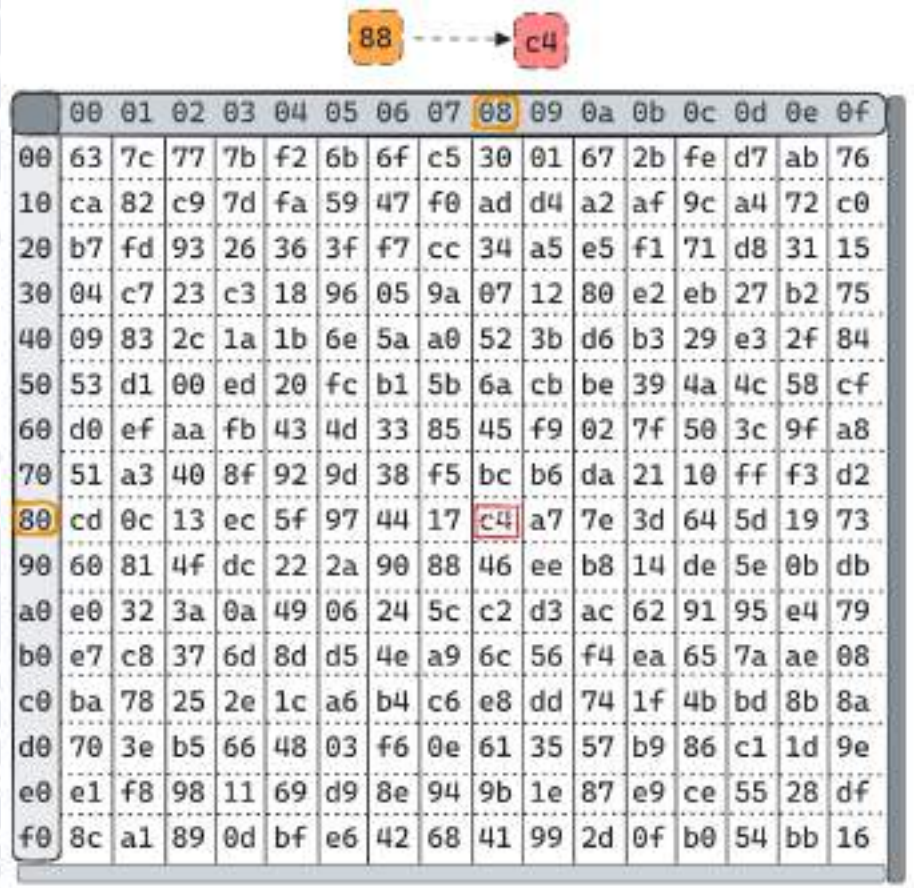


Рисунок 1.4 – Приклад шифрування символів на бінарному рівні

Далі розглянемо основні раунди AES. Кожен раунд складається з чотирьох різних етапів:

I. “SubBytes”. Цей крок є нелінійним процесом заміни, під час якого кожен байт у матриці стану перетворюється за допомогою спеціально розробленої таблиці, відомої як S-бокс (рис. 1.5). S-бокс сконструйований таким чином, щоб ефективно протистояти різним криптографічним атакам. Наприклад, байт 'A' згідно із S-боксу може бути замінений на байт 'B' (рис 1.6).



	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Рисунок 1.5 – Приклад S-box таблиці

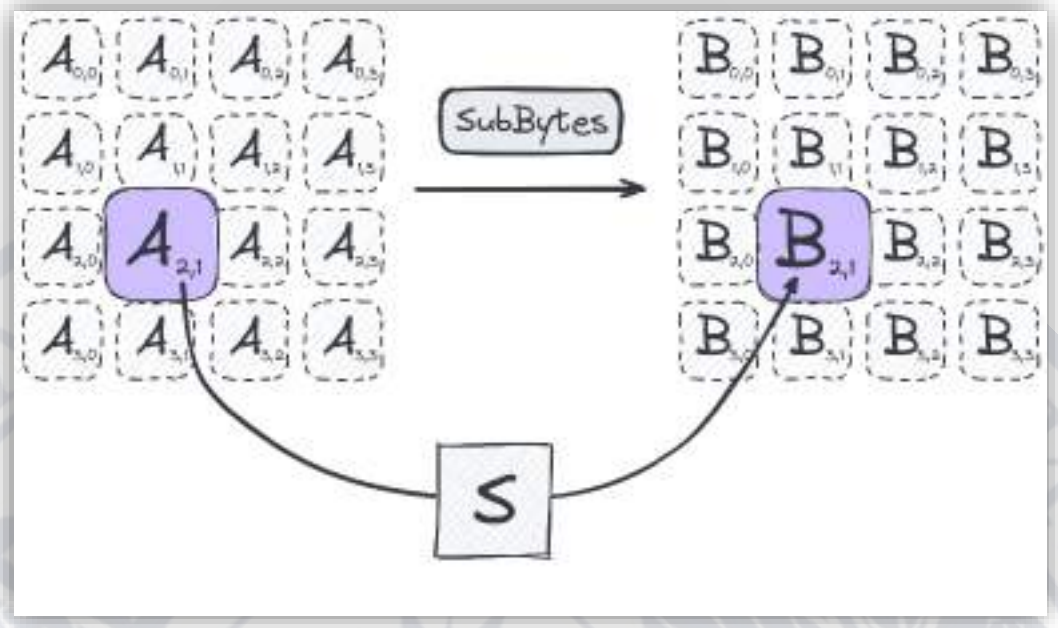


Рисунок 1.6 – Приклад операції заміни із використанням таблиці S-box

II. “ShiftRows”. На цьому етапі відбувається крок транспозиції, де рядки стану циклічно зсуваються [14]. Перший рядок не зсувається, другий рядок зсувається на одну позицію вліво, третій – на дві позиції, а четвертий – на три позиції (рис 1.7).

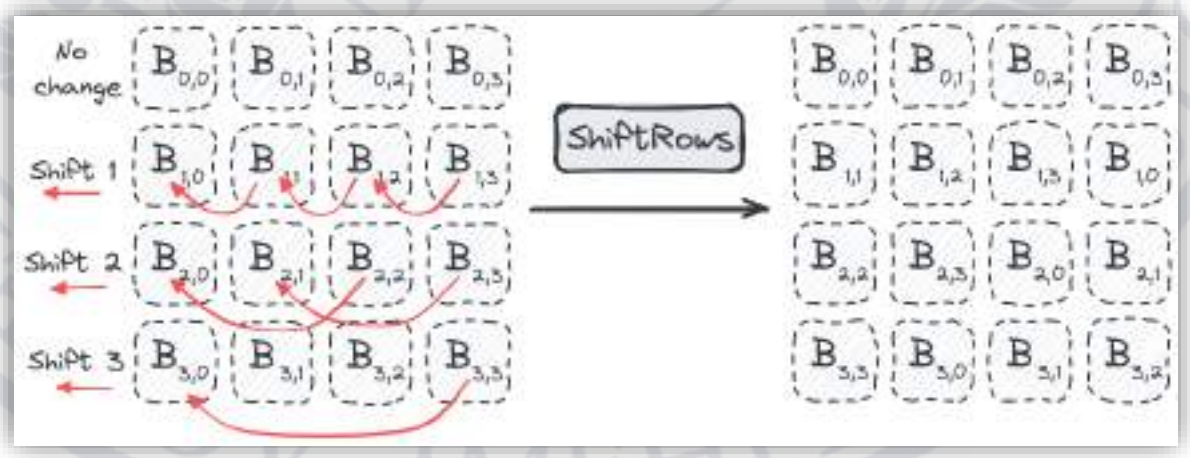


Рисунок 1.7 – Приклад операції транспозиції рядків матриці

III. “MixColumns”. На етапі “MixColumns” кожна колонка стану трансформується з використанням певної математичної операції (рис. 1.8). Ця операція здійснюється у спеціалізованому математичному полі, де кожна колонка розглядається як поліном (рис. 1.9). Здійснюється множення поліномів за модулем заданого фіксованого полінома, що дозволяє ефективно розподілити вхідні дані серед колонок та підвищити розсіювання даних у шифрі [14].

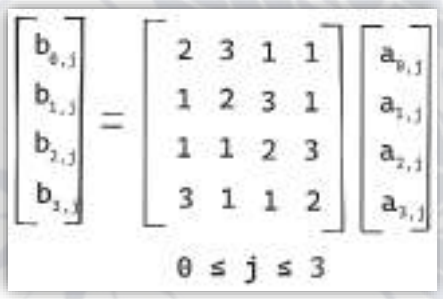


Рисунок 1.8 – Приклад використання фіксованої матриці

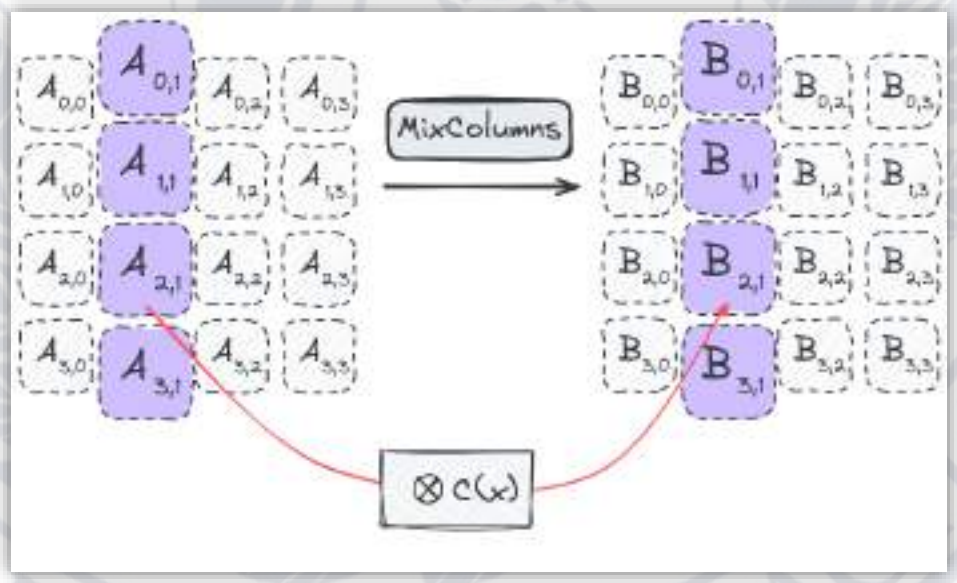


Рисунок 1.9 – Перетворення кожного стовпця матриці використовуючи фіксований поліном

IV. “AddRoundKey”. На цьому етапі, подібно до початкового раунду, раундовий ключ (отриманий з розширення основного ключа) комбінується

з матрицею стану через операцію XOR (рис. 1.10). Цей крок вводить ключ у структуру даних стану, забезпечуючи додатковий рівень складності та посилення захисту [14].

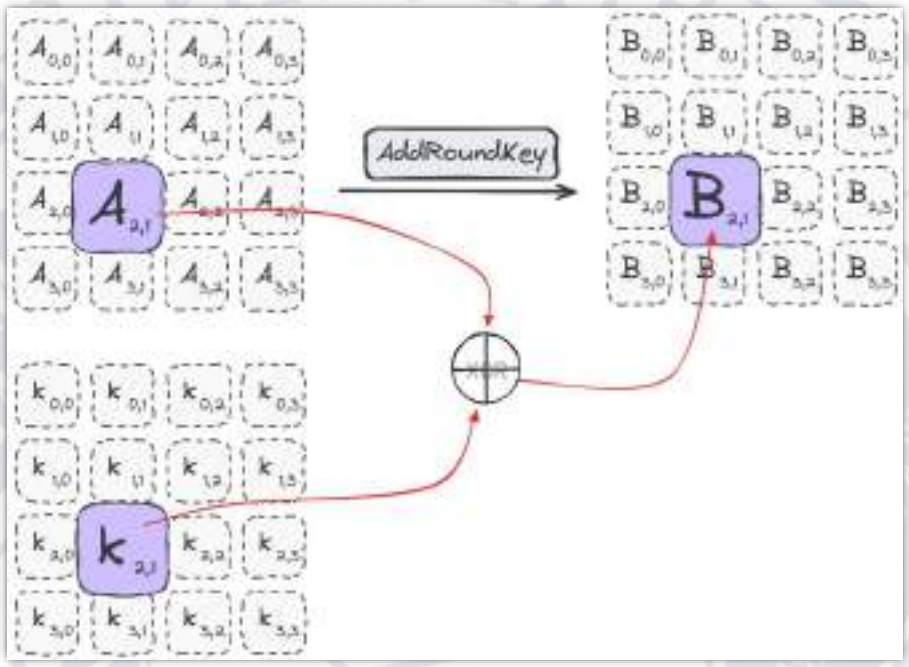


Рисунок 1.10 – Приклад комбінування матриці станів та раундового ключа

Останній раунд шифрування AES включає ті ж кроки, що і основні раунди, але без етапу “MixColumns”. Це виключення оптимізує процес шифрування та готує дані до виведення. Остання матриця стану потім перетворюється назад у простий набір символів [14].

1.3.2 Різновиди режимів роботи алгоритму AES: порівняння ECB, CBC, CFB, OFB

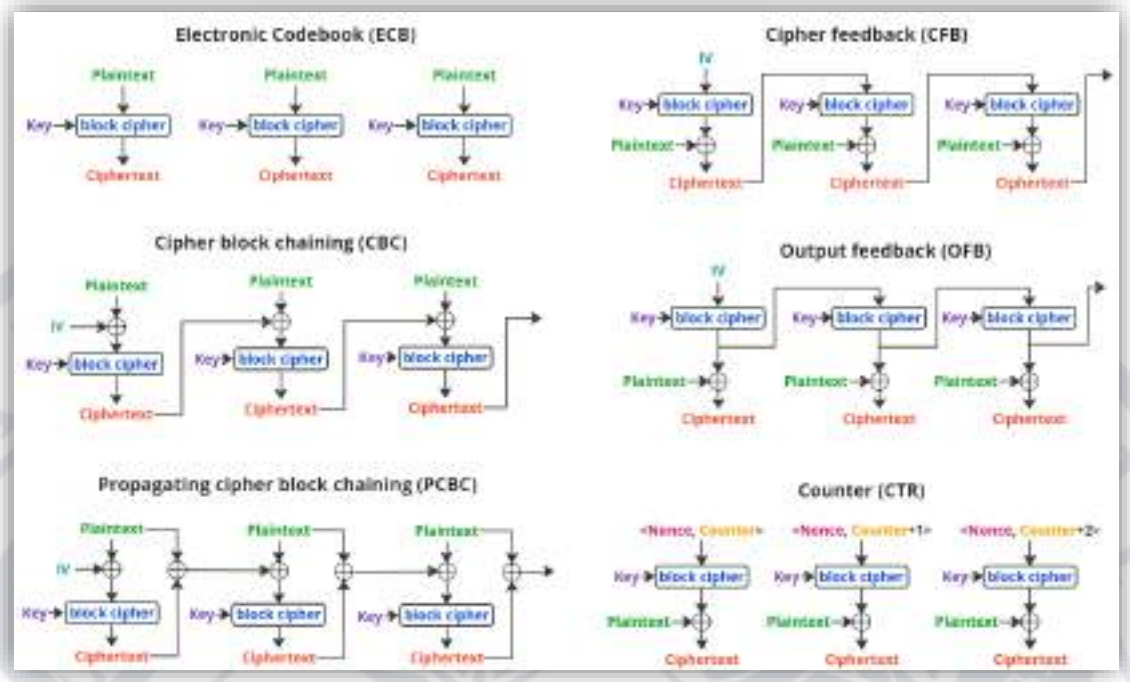


Рисунок 1.11 – Шість загальних режимів роботи блокового шифру для шифрування [15]

ECB (Електронна Кодова Книга): В цьому режимі текст розбивається на блоки, кожен з яких шифрується незалежно від інших [15]. Хоча ECB є простим у використанні та дозволяє шифрувати блоки паралельно, що підвищує швидкість обробки, він має обмеження при роботі з великими обсягами даних або текстом з повтореннями, оскільки може утворювати впізнавані шаблони в шифрованому тексті.

CBC (Ланцюгове Шифрування Блоків): У цьому режимі кожен блок даних перед шифруванням комбінується з попередньо зашифрованим блоком. Перший блок шифрується з використанням вектора ініціалізації [15]. CBC забезпечує високий рівень безпеки, оскільки складно аналізувати послідовності блоків, але вимагає високої точності передачі даних, оскільки помилки можуть поширюватися на наступні блоки.

Режим CTR (Counter mode) перетворює блочний шифр у потоковий, використовуючи лічильник і унікальне число (nonce) для генерації ключового потоку [15]. Цей режим дозволяє паралельне шифрування та розшифрування, що

робить його ідеальним для використання в мультипроцесорних системах. При цьому необхідно забезпечити ретельне управління лічильниками та nonce для збереження безпеки.

OFB та CFB (Зворотний Зв'язок та Зворотний Зв'язок Шифру): Ці режими шифрування створюють ключовий потік для шифрування даних, забезпечуючи, що помилки у передачі не впливають на наступні блоки [15]. Однак ці методи вимагають послідовної обробки, що може сповільнювати загальну швидкість обробки.

GCM (Режим Галуа/Лічильника): Цей режим інтегрує шифрування з криптографічною хеш-функцією, що забезпечує одночасну конфіденційність та цілісність даних [15]. Висока продуктивність досягається завдяки можливості паралельної обробки, однак вимагає уважного управління ключами та параметрами безпеки.

Для побудови Python додатку, розробка якого буде розглянута у наступних розділах, буде використане ланцюгове шифрування блоків (CBC) так як воно найкраще відповідає вимогам завдання та забезпечує швидкість та конфіденційність обробки даних.

1.4 Аналоги та успішні рішення на ринку

Розширений стандарт шифрування широко використовується у різних протоколах безпеки та додатках через його надійність та стійкість. Він є вибором для захисту мереж Wi-Fi за стандартом WPA2, забезпечуючи приватність даних, що передаються через бездротові з'єднання. У сфері віртуальних приватних мереж (VPN), AES допомагає у захисті інтернет-трафіку, оберігаючи дані, які проходять через потенційно ненадійні мережі. Його прийняття урядовими та фінансовими секторами для захисту класифікованої та чутливої інформації підкреслює його критичну роль у світових цифрових рамках безпеки.

Що до сфери шифрування даних, то аналоги, як TrueCrypt та VeraCrypt, досягли значних успіхів. TrueCrypt, колись основний інструмент шифрування з

відкритим вихідним кодом, припинив свою розробку у 2014 році, що змусило користувачів шукати альтернативні рішення, такі як VeraCrypt. VeraCrypt, який є наступником TrueCrypt, пропонує покращені заходи безпеки, але також має підвищену складність, включно з переходом до більш закритого вихідного коду. Ця зміна викликала занепокоєння серед користувачів, оскільки неможливість доступу до коду унеможливлює перевірку його цілісності та викликає підозри щодо потенційних вразливостей чи вбудованих шпигунських програм [16].

На цьому тлі, ідея щодо додатку, який буде представляти сучасний та гнучкий підхід до шифрування набуває особливої актуальності. Використовуючи міцні стандарти та забезпечуючи повну прозорість із легкістю модифікацій, можна ефективно вирішити обмеження попередників.

В першому розділі були розглянуті теоретичні відомості про те що таке шифрування, які існують алгоритми та як і навіщо їх застосовувати. Також було проведено детальний опис послідовності та етапів перетворення тексту на набір символів використовуючи ланцюгове шифрування блоків. Далі, даний алгоритм буде використано у якості ядра майбутнього Python додатку.

РОЗДІЛ 2
РЕАЛІЗАЦІЯ PYTHON ПРОГРАМИ НА ОСНОВІ ADVANCED ENCRYPTION STANDARD

2.1 Опис програми та її функцій

Під час розробки основних функцій додатку, потрібно приділити найвищий пріоритет створенню адаптивного інтерфейсу користувача. Адаптивний інтерфейс є ключовим елементом, який забезпечує винятковий користувацький досвід, об'єднуючи простоту управління з сучасним дизайном. Філософія дизайну акцентує увагу на мінімалізмі та інтуїтивності, реалізованій через використання зрозумілих кнопок та текстових полів, що спрощують взаємодію з додатком і гарантують високий рівень зручності користувачів.

Переходячи до технічних особливостей, додаток буде включати в себе інструмент шифрування файлів. Враховуючи необхідність забезпечення високої безпеки при зберіганні та передачі даних, буде використано спеціалізований алгоритм шифрування, який оптимально підходить для таких завдань. Цей алгоритм гарантує швидку та надійну обробку даних, а також підтримку складних паролів, що можуть бути довшими за стандартні розміри блоків 128-біт, 192-біт, і 256-біт, які використовуються в шифруванні за стандартом AES.

Для підвищення безпеки буде інтегровано механізм використання "солі" (salt), який передбачає використання випадково згенерованих значень, що значно зменшують ризик використання "Rainbow table" для підбору паролів [17]. Ця технологія додає додатковий рівень захисту, роблячи паролі набагато важчими для злому.

Окрім того, з огляду на ризики, пов'язані зі збереженням чутливих даних, як наприклад, паролів у звичайних текстових файлах, додаток також буде пропонувати рішення для їх шифрування. Це дозволить користувачам зберігати свою інформацію в зашифрованому та безпечному форматі, запобігаючи її витоку та забезпечуючи конфіденційність.

2.2 Вибір технологій для реалізації GUI та шифрування

Перед початком розробки проекту, перш за все, важливо обрати платформу, на якій буде побудована система. Сучасний світ програмування пропонує широкий спектр можливостей, включаючи різноманітні мови програмування, фреймворки та бібліотеки, які розробляються та підтримуються численними компаніями. Одним з найбільш ефективних варіантів для реалізації проекту є мова програмування Python, яка забезпечує виконання всіх необхідних вимог.

Python є простою та універсальною мовою програмування, створеною Гвідо ван Россумом у 1991 році [18]. Ця мова підтримує кілька парадигм програмування, що дозволяє їй залишатися гнучкою та адаптивною до різних проектних вимог. Особливістю Python є його багата стандартна бібліотека, яка значно спрощує процес кодування і знижує загальну складність розробки. Простий синтаксис мови і активна спільнота користувачів сприяють її популярності як серед початківців, так і серед досвідчених розробників.

Python відрізняється простотою у використанні, що значно прискорює процес розробки додатків. Завдяки наявності великої кількості готових бібліотек, розробники можуть швидко інтегрувати необхідні функціональні можливості, порівнюючи процес зі складанням конструктора. Використання таких бібліотек є аналогічним відвідуванню бібліотеки з книгами, де ми можемо взяти існуючий контент і адаптувати його під наші потреби. Це не тільки прискорює процес розробки, а й дозволяє розробникам зосередитись на створенні унікальних особливостей проекту, не витрачаючи час на розробку вже існуючих рішень. Так само, як бібліотека в світі книг надає доступ до знань, бібліотека програмування пропонує колекцію попередньо написаного коду, що може бути використаний у різних проектах. Це значно спрощує процес програмування, оскільки не потрібно створювати весь код з нуля кожен раз, коли починаєш новий проект, що робить Python ідеальним вибором для нашого проекту, дозволяючи виконати всі поставлені цілі ефективно і вчасно.

Коли мова йде про розробку графічних інтерфейсів користувача (GUI) у Python, розробники мають великий вибір бібліотек, кожна з яких має свої особливості, зручність використання та характеристики продуктивності. Розглянемо декілька популярних бібліотек для розробки GUI:

- Tkinter - це стандартна бібліотека, яка включена в стандартний пакет, що робить її легкодоступною та простою [19]. Вона ідеально підходить для маленьких проектів. Проте, через її прості візуальні можливості, функцій Tkinter не вистачає для створення складних чи візуально привабливих інтерфейсів.
- PyQt - бібліотека, яка є однією з найбільш потужних у арсеналі. Вона підтримує багато платформ, включаючи Windows, macOS, та Linux, і забезпечує інтуїтивний вигляд на кожній з них. PyQt включає інструменти, такі як Qt Designer, який дозволяє легко створювати дизайн інтерфейсу методом перетягування компонентів [20].

  PySide6 - офіційний набір прив'язок Python до фреймворку Qt 6, що дозволяє розробляти багатофункціональні та динамічні графічні інтерфейси користувача [21]. Ця бібліотека підтримує основні операційні системи, гарантуючи єдиний вигляд на різних платформах. PySide6 включає широкий спектр інструментів, які допомагають створювати від простих додатків до великих корпоративних проектів.

Щодо бібліотек, які допомагають з шифруванням AES CBC у Python, то серед доступних опцій є кілька добре відомих та ефективних рішень, кожне з яких має свої переваги та особливості:

- PyCrypto - це одна з найстаріших бібліотек для криптографії. Вона включає підтримку багатьох алгоритмів, включаючи AES у режимі CBC. Вона дозволяє розробникам контролювати деталі шифрування, такі як розмір

блоку, ключа та початкового вектору. Однак варто зазначити, що PyCrypto не підтримується з 2018 року, що може призвести до проблем з безпекою.

- PyCryptodome - це форк PyCrypto, який додає покращену функціональність та підтримку сучасних криптографічних вимог. PyCryptodome сумісний із PyCrypto, тож розробники можуть легко перейти на більш безпечний варіант без зміни коду. Вона також підтримує AES CBC і забезпечує сучасніший і безпечніший підхід до шифрування.
- Cryptography - це сучасна бібліотека, яка фокусується на забезпеченні легкого доступу до криптографічних функцій із високим рівнем абстракції. Вона підтримує шифрування AES CBC з автоматичним управлінням ключами та початковими векторами, роблячи процес більш зрозумілим та безпечним для розробників.

2.2.1 Переваги PySide6 для створення GUI

Одна з ключових особливостей PySide6 - це використання Qt Designer, інструменту, який допомагає проектувати інтерфейси з технікою "перетягування", що спрощує розробку. Різниця між PySide6 і PyQt полягає в тому, що він доступний під ліцензією LGPL, яка є більш гнучкою для комерційного використання, оскільки не вимагає відкриття коду додатку на тих самих умовах. Ця гнучкість у ліцензуванні, поєднана з потужними функціями, робить PySide6 ідеальним вибором для подальшої розробки.

PySide6 - це Python доповнення до фреймворку Qt6, який є одним з наймогутніших і популярних кросплатформених фреймворків розробки додатків. Він розширює можливості свого попередника, Qt5, забезпечуючи кращу продуктивність і більш сучасні функціональні можливості.

Однією зі значних переваг PySide6 є його інтеграція з Qt Quick і QML (Qt Modeling Language) [21]. Qt Quick - це стандарт для створення плавних, анімованих та приємних для очей GUI, які добре працюють як на настільних, так і на мобільних пристроях. За допомогою QML розробники можуть проектувати

інтерактивні користувацькі інтерфейси з меншою кількістю коду та складнощів. QML також дозволяє дизайнерам і розробникам працювати разом, роблячи робочий процес швидшим і ефективнішим.

Крім того, PySide6 підтримує розширену графіку з інтеграцією OpenGL, що дозволяє створювати додатки з багатим графічним вмістом, включаючи складні анімації та ефекти. Це особливо корисно для додатків, які вимагають високого рівня візуальної естетики або тих, що є графічно інтенсивними, наприклад, ігор або інструментів дизайну. Він включає підтримку 3D-графіки через модуль Qt 3D, який можна використовувати для створення 3D-віджетів і обробки 3D-вмісту [21]. Це робить PySide6 підходящим вибором для розробки складних CAD-додатків, наукових візуалізацій або будь-якого програмного забезпечення, що вимагає 3D-графіки.

Крім своїх потужних можливостей GUI, PySide6 пропонує широкий спектр інструментів для мережі, з'єднань і управління базами даних. Він включає класи для управління HTTP-комунікаціями, доступ до SQL-баз даних та обробку XML та JSON-даних [21]. Цей комплексний набір інструментів дозволяє розробникам будувати не тільки автономні додатки, а й складні, орієнтовані на дані системи, інтегровані з веб-сервісами та базами даних.

Архітектура, орієнтована на події, забезпечує, що додатки є одночасно адаптивними та продуктивними. Фреймворк ефективно обробляє взаємодії користувачів, системні події та інші вхідні дані, дозволяючи розробникам зосередитися на основній функціональності своїх додатків, не турбуючись про низькорівневі деталі управління подіями та обробки GUI.

PySide6 це не просто про створення простих GUI-додатків, а про створення сучасних, складних програмних систем, які можуть функціонувати на різних платформах. Його обширна бібліотека, сумісність з різними медіа і форматами даних, а також можливість інтеграції з іншими бібліотеками Python та API роблять його потужним інструментом.

2.2.2 Роль "cryptography.fernet" у реалізації алгоритму AES

Серед перелічених бібліотек шифрування бібліотека Cryptography виділяється як найкращий вибір, особливо при розробці додатків, які вимагають міцного криптографічного захисту. Вона створена для високорівневого симетричного шифрування за допомогою алгоритму AES у режимі CBC з 128-бітним ключем [22]. Цей вибір забезпечує міцну систему безпеки, придатну для більшості додатків, які потребують шифрування. Fernet побудований на принципах, які віддають перевагу безпеці та простоті, що робить його відмінним інструментом для розробників, яким потрібно впроваджувати шифрування, не вдаючись до глибоких деталей криптографії.

Однією з ключових особливостей Cryptography.fernet є його простота у використанні. API простий, з основними функціями, як шифрування та розшифрування, доступними для розробників [22]. Ця простота не йде на шкоду безпеці, оскільки вона використовує комбінацію криптографічних технік для забезпечення цілісності та конфіденційності даних. Кожне повідомлення, зашифроване за допомогою Fernet, включає мітку часу, яка використовується для перевірки моменту шифрування даних, додаючи ще один рівень безпеки за допомогою терміну дії токена.

Сильні сторони Cryptography.fernet численні. По-перше, вона автоматизує обробку ключів шифрування, що може бути складним аспектом безпечного впровадження шифрування. Бібліотека генерує ключі, які безпечно зберігати та використовувати протягом тривалого часу, зменшуючи ризик витоку ключів. По-друге, включення мітки часу в процес шифрування не тільки допомагає управляти життєвим циклом токенів, але й захищає від певних типів криптографічних атак, таких як атаки повторного відтворення.

Розробляючи рішення з шифруванням за допомогою Cryptography.fernet, розробники можуть створювати безпечні системи комунікацій, безпечно зберігати дані користувачів та забезпечувати цілісність переданих інформацій. Наприклад, додаток, що зберігає чутливу інформацію користувачів, таку як

особисті документи або паролі, може використовувати Fernet для шифрування цих даних перед їх зберіганням, забезпечуючи, що навіть у разі несанкціонованого доступу до носія даних, дані залишаються захищеними [22].

Крім того, Cryptography.fernet підтримує ротацію ключів, ключову функцію для підтримки безпеки в довгострокових додатках. Цей процес включає генерацію нового ключа та забезпечення того, що будь-які дані, раніше зашифровані, можуть бути розшифровані за допомогою старого ключа, а потім перешифровані новим. Цей метод допомагає зменшити ризики, пов'язані з потенційним компрометуванням ключа з часом.

Бібліотека Cryptography, включаючи її модуль Fernet, доповнена обширною документацією та спільнотою розробників, які сприяють її постійному вдосконаленню [22]. Ця спільнота та наявність ресурсів полегшують впровадження та виправлення неполадок, що додатково підвищує її привабливість для розробників.

2.3 Пошук платформи для збереження та керування проектом

На даний момент платформи як GitHub та GitLab відіграють важливу роль у розробці будь якого софту, надаючи інструменти, які допомагають створювати, зберігати, змінювати та керувати кодом додатків. Ці платформи дуже корисні, бо пропонують централізоване місце для співпраці команд над проектами, відстеження змін і управління різними версіями їхнього коду.

GitHub є однією з найпопулярніших платформ серед розробників. Він відомий своїми потужними функціями, що підтримують спільні проекти, такі як відстеження помилок, запити на додавання функцій, управління завданнями та вікі для документації проектів [23]. Одна з найбільших переваг використання GitHub — це його велика спільнота, що полегшує співпрацю з іншими розробниками по всьому світу. Проте через велику популярність він іноді може працювати повільно, коли ним одночасно користується багато людей. Також, хоча

GitHub надає приватні репозиторії безкоштовно, існують обмеження на кількість співпрацівників у приватних проектах для безкоштовних акаунтів.

З іншого боку, GitLab також є дуже потужною платформою з подібними функціональними можливостями до GitHub. Він відрізняється наявністю вбудованих інструментів неперервної інтеграції та неперервного розгортання (CI/CD), які дуже корисні для автоматизації тестування та розгортання коду [24]. Це може значно заощадити час і зменшити помилки під час процесу розробки. Крім того, GitLab забезпечує більш всебічний контроль над дозволами, що може бути особливо корисним для більших команд або організацій, яким потрібні детальні налаштування доступу. Однак інтерфейс GitLab може бути трохи складнішим для навігації, що може стати викликом для нових користувачів.

Для керування розробкою нашого додатка ми будемо використовувати GitHub разом з Git, який є системою контролю версій. Git дозволяє нам відстежувати зміни у нашому коді та ефективно співпрацювати з іншими [25]. Щоб забезпечити організовану розробку та ефективне керування різними частинами нашого проекту, ми створимо окремі гілки у нашому репозиторії. Зокрема, у нас буде одна гілка, присвячена користувацькому інтерфейсу (UI), інша — бізнес-логіці, а третя — алгоритму шифрування. Це розділення допоможе плавніше керувати процесом розробки і дозволить зосередитися на конкретних задачах окремо.

2.4 Вибір open source ліцензії та створення README.md

Для проекту вирішено використовувати бібліотеку «PySide6», яка доступна під різними ліцензіями. Після ретельного розгляду, обрання загально публічної ліцензії (GPLv2) відповідає нашій меті зробити проект відкритим, дозволяючи іншим вільно використовувати, змінювати та розповсюджувати програму, а також вносити свій вклад у її удосконалення та розвиток.

GPLv2 є однією з найпопулярніших ліцензій відкритого коду і відома своїми суворими вимогами до модифікацій. Це означає, що будь-яке програмне

забезпечення, яке похідне від нашого проекту або створене з його використанням, також має бути випущене під цією ж ліцензією, GPLv2 [26]. Ця вимога допомагає забезпечити, що свободи, надані ліцензією, поширюються на всі версії програми. Однак важливо зауважити, що це також може бути розглянуто як обмеження, оскільки це обмежує використання програми у комерційних проектах, які не мають наміру випускати свій вихідний код.

Вибір GPLv2 має кілька переваг. Вона сприяє створенню певного середовища, де розробники можуть вносити вклад у роботу один одного і допомагати удосконалювати програмне забезпечення. Також вона забезпечує правовий захист для оригінальних розробників та гарантує, що вихідний код залишається доступним і вільним для всіх. Однак вимога ділитися покращеннями може відштовхнути деякі комерційні структури від використання програми, оскільки вони можуть не бажати розкривати свої модифікації.

Додатково, щоб забезпечити максимальну доступність та зрозумілість проекту, створимо файл «README.md» файл, який слугуватиме як перша документація, з якою зіткнуться користувачі та учасники. У файлі «README.md» буде включено всебічний огляд проекту, включаючи його мету, інструкції з налаштування та використання. Також там буде розділ, присвячений поясненню вибору ліцензії, конкретно обрано GPLv2 і що це означає для користувачів та учасників.

2.5 Етапи реалізації програми та їх опис

Процес розробки буде структурований у чіткі, послідовні етапи, щоб забезпечити функціональність та зручність використання. Спочатку зосередимося на реалізації та ретельному тестуванні алгоритмів шифрування, щоб переконатися, що вони безпечні та ефективні. Після цього ми використаємо QT Designer для проектування графічного інтерфейсу, що допоможе нам візуально розмістити компоненти та екрани, необхідні для програми [27]. Як тільки дизайн буде готовий, приступимо до написання фактичного коду

інтерфейсу, інтегруючи його з підкладковою логікою Python. На завершення, підключимо алгоритми шифрування до інтерфейсу, активуємо інтерактивні функції та проведемо всеосяжні тести, щоб забезпечити безперебійну роботу всіх компонентів разом.

2.5.1 Побудова архітектури

Архітектура Python-програми розроблена таким чином, щоб бути модульною та легкою у керуванні, використовуючи PySide6 для створення графічного інтерфейсу користувача. Основна структура зосереджена навколо кількох ключових файлів, які керують різними аспектами програми, забезпечуючи, що кожна частина має чітко визначену відповідальність, дотримуючись таким чином принципу розділення обов'язків.

Main.py:

Цей файл є точкою входу для програми. Він містить функцію main(), де починається виконання програми. Тут потік програми ініціюється шляхом створення екземпляру Qapplication, який є необхідним для будь-якого застосунку PySide6, оскільки він керує потоком управління графічним інтерфейсом користувача та основними налаштуваннями. Потім створюються екземпляри класів AppController та UserInterface. UserInterface отримує екземпляр AppController, зв'язуючи користувацький інтерфейс з логікою програми. Головне вікно відображається, і цикл обробки подій програми починається з app.exec(), чекаючи на взаємодії користувача, такі як кліки та вводи.

App_controller.py:

Клас AppController, який походить від Qobject, служить посередником між користувацьким інтерфейсом та криптографічним сервісом. Він виконує завдання, такі як перевірка помилок та забезпечення правильної передачі даних між UI та бекенд-логікою. Цей клас ефективно керує комунікацією, переконуючись, що дії користувача на інтерфейсі викликають правильні реакції в бекенді.

User_interface.py:

У файлі UserInterface знаходяться всі елементи, з якими безпосередньо взаємодіє користувач, такі як кнопки, текстові поля та інші елементи управління. Він також обробляє події, що спричинені діями користувача, такими як натискання кнопок або відправлення форм. Цей файл має важливе значення для захоплення вводу користувача та відображення інформації.

Ui_main_window.py:

Цей модуль містить клас Ui_MainWindow, який відповідає за конфігурацію зовнішнього вигляду головного вікна, включаючи налаштування кольорів, шрифтів та розмірів. Ця конфігурація є важливою, оскільки безпосередньо впливає на зручність і естетику програми, роблячи взаємодію користувача або приємною, або складною.

Crypto_service_utility.py:

Наостанок, CryptoServiceUtility — це клас, що реалізує основну функціональність шифрування/розшифрування текстів та файлів. Він включає методи для шифрування текстів, розшифрування текстів, генерації ключів шифрування з випадковими солями та обробку файлів. Даний сервіс є фундаментальним для безпечної обробки даних, забезпечуючи, що вся чутлива інформація, яка обробляється програмою, належним чином захищена.

Архітектура розроблена так, щоб бути міцною, але достатньо гнучкою для змін та розширень. Кожен компонент створений з певною роллю, забезпечуючи організованість та можливість підтримки програми.

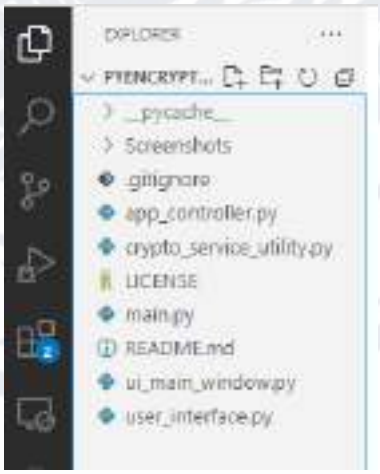


Рисунок 2.1 – Лістинг файлів додатку

2.5.2 Створення основного вікна програми

При розробці головного вікна програми обреремо простий, але сучасний дизайн. Інтерфейс буде мати заспокійливі сині кольори та прозорий фон, що забезпечує візуально привабливий та інтуїтивно зрозумілий користувацький досвід. Вікно організоване з кількома функціональними вкладками, серед яких "Encryption" (Шифрування) та "Decryption" (Розшифрування), що дозволяє користувачам легко орієнтуватися в програмі залежно від їхніх потреб.

Стандартний вигляд — це вкладка "Text Encryption" (Шифрування тексту), де користувачі бачать три текстові поля: одне для вводу тексту, який потрібно зашифрувати, одне для пароля, та одне для відображення зашифрованого результату. Також є прапорець, який дозволяє користувачам вибирати, чи показувати пароль для підвищення безпеки.

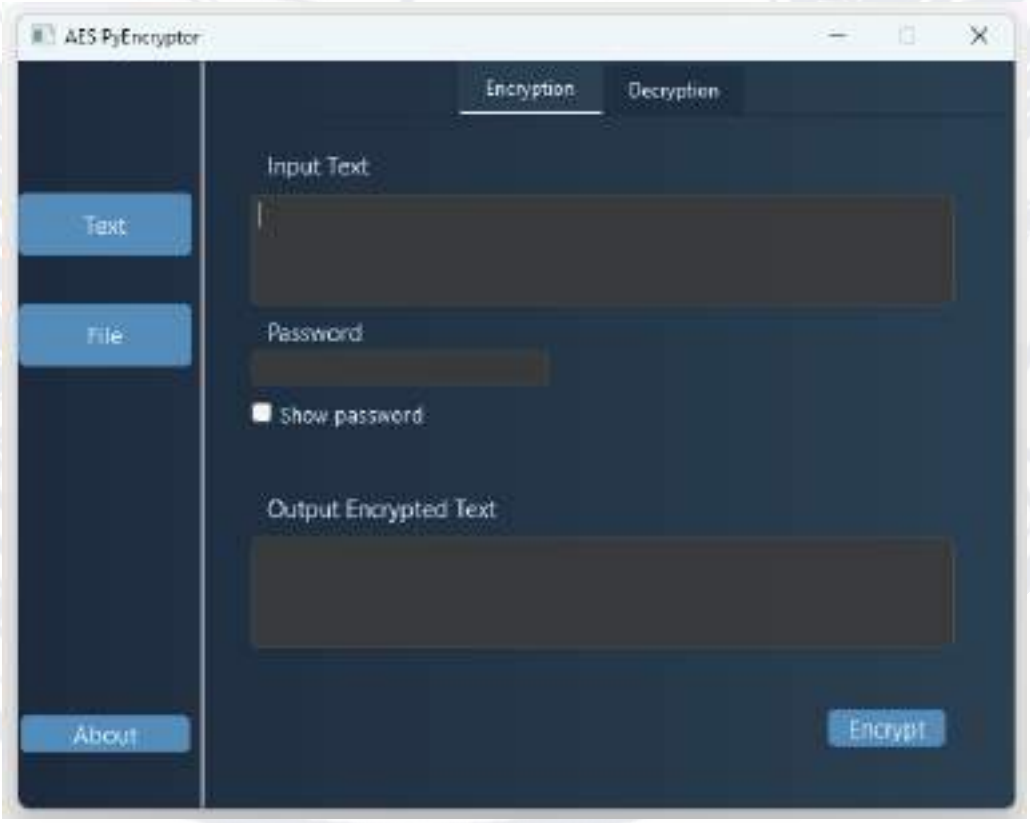


Рисунок 2.2 – Вигляд головного вікна додатку

Натискаючи на кнопку "File" у вкладці, інтерфейс змінюється, щоб вмістити шифрування файлів. У цьому режимі користувачі бачать індикатор прогресу та текстове поле для шляху до файлу, поруч з кнопкою для вибору файлу, який вони хочуть зашифрувати або розшифрувати. Також є текстове поле, де користувачі можуть ввести нове ім'я файлу та вибрати розширення файлу, щоб зашифрований файл був збережений належним чином. Ця секція також включає текстове поле для пароля та велику кнопку для початку процесу шифрування чи розшифрування.

Крім того, у головному вікні є кнопка "About" (Про програму). Натискаючи цю кнопку, відкривається додаткова інформація про програму, включно з QR-кодом, пов'язаним з нашим обліковим записом GitHub. Цей QR-код дозволяє користувачам легко доступити та ознайомитися з кодом програми, сприяючи прозорості та довірі до практик розробки та безпеки програмного забезпечення.

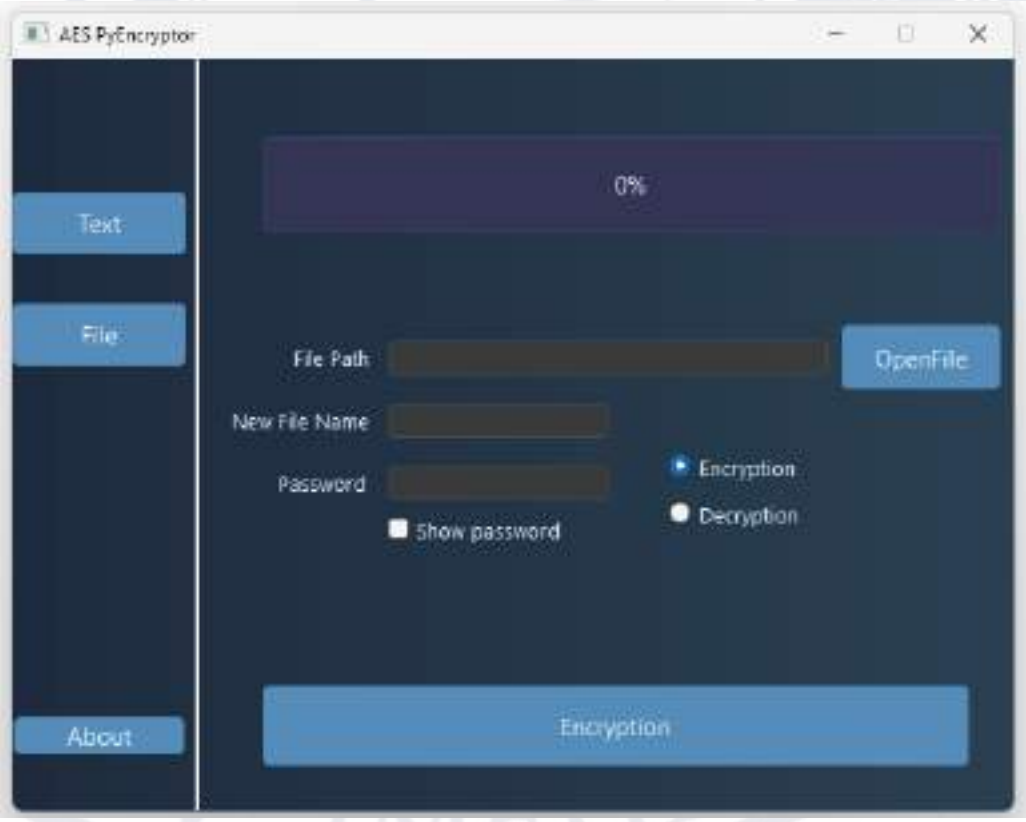


Рисунок 2.3 – Вигляд вікна для шифрування файлів

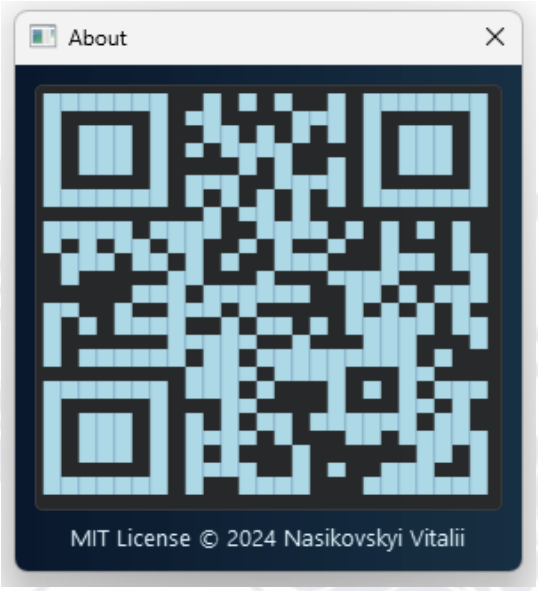


Рисунок 2.4 – Вигляд вікна “Про програму”

Щоб підвищити зручність користувачів і забезпечити надійну роботу програми, наша програма включає спеціальне вікно "Помилка". Ця функція спеціально розроблена, щоб сповіщати користувачів у разі будь-яких проблем під час процесу шифрування.

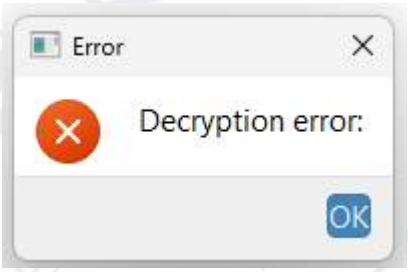


Рисунок 2.5 – Вигляд вікна “Помилка”

2.5.3 Реалізація функціоналу шифрування та дешифрування

Створимо клас “CryptoServiceUtility” (рис. 2.6), який буде включати основний набір інструментів для шифрування та розшифрування за допомогою симетричного ключа криптографії. Цей клас дозволить користувачам безпечно шифрувати та розшифровувати тексти та файли, використовуючи пароль.

```python
class CryptoServiceUtility:
    @staticmethod
    def generate_key(password):
        """Generate a 32-byte encryption key from the provided user password."""
        salt = os.urandom(16)  # Generate a random 16-byte (128-bit) salt
        kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=32,  # Derive a 32-byte (256-bit) key
            salt=salt,
            iterations=480000,  # Number of iterations for the PBKDF2 algorithm
        )
        key = base64.urlsafe_b64encode(kdf.derive(password.encode()))
        return key, salt

    @classmethod
    def encrypt_text(cls, text, password):
        """Encrypt text."""
        key, salt = cls.generate_key(password)
        cipher_suite = Fernet(key)
        encrypted_text = cipher_suite.encrypt(text.encode())  # Encrypt the text
        # Prepend the salt to the encrypted data before converting to hex
        return (salt + encrypted_text).hex()
```

Рисунок 2.6 – Лістинг коду генерації ключа та шифрування тексту

Спочатку розглянемо, як генеруються ключі шифрування. Метод “generate_key” буде дуже важливим, оскільки він буде створювати 32-байтові ключі шифрування із паролів, наданих користувачем. Процес починається з генерації випадкової 16-байтної “солі” за допомогою “os.urandom”. Ця “сіль” підвищує безпеку криптографічного процесу, забезпечуючи унікальність виводу функції похідного ключа (KDF), навіть коли один і той самий пароль використовується кілька разів. Для виводу ключа використовується алгоритм PBKDF2 з хеш-функцією SHA-256. PBKDF2 розрахований на те, щоб зробити зменшити ймовірність підбору паролю за допомогою атаки brute-force. У цій реалізації алгоритм виконує 480 000 ітерацій, що значно підвищує безпеку, збільшуючи час обчислень, необхідний для виведення ключа.

Зашифрований ключ разом з сіллю потім використається для шифрування текстів та файлів. У методі “encrypt_text”, після генерації ключа, буде створюватись екземпляр класу Fernet. Fernet — це метод симетричного шифрування, який забезпечує, що дані, раз зашифровані, можуть бути розшифровані до своєї первісної форми тільки за допомогою того ж ключа.

Вхідний текст шифрується, а "сіль" додається до цих зашифрованих даних перед перетворенням усіх даних в шістнадцятковий рядок для зберігання або передачі.

Що до процесу шифрування файлів, то саме для нього створимо окремий метод “encrypt_file”. Єдина різниця між “encrypt_text” та “encrypt_file” – це те, що дані файлу потрібно перетворити в текстовий рядок. Метод буде зчитувати бінарні дані з файлу, шифрувати їх, а потім записувати "сіль" та зашифровані дані назад у інший файл із іншим розширенням.

Для процесу розшифрування тексту створимо метод “decrypt_text”, який буде приймати зашифрований текст як вхід, після чого буде витягувати “сіль”, використану під час шифрування, та використовуючи її разом з паролем для відтворення ключа шифрування.

Для файлів метод “decrypt_file” слідує тим самим принципам. Він читає зашифрований файл, щоб отримати "сіль" та зашифровані дані, відновлює ключ шифрування, а потім розшифровує дані, зберігаючи їх у новому файлі.

В другому розділі було розглянуто технології, за допомогою яких можна реалізувати необхідний функціонал додатку. Описано основні бібліотеки та потенційні платформи для керування проектом. Створено основні етапи розробки, які включають побудову архітектури, реалізацію дизайну та написання основних методів шифрування.

РОЗДІЛ 3
ПРАКТИЧНЕ ЗАСТОСУВАННЯ ПРОГРАМИ ДЛЯ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ДАНИХ

3.1 Приклади використання програми у реальних сценаріях

Один з поширених випадків, де шифрування є ключовим, - це захист особистих даних. Людям часто потрібно забезпечувати безпеку чутливої особистої інформації, такої як фінансові документи, дані облікових записів та приватне спілкування. Наприклад, особа може використовувати цей додаток для шифрування особистих податкових документів перед їхнім зберіганням у хмарному сервісі. Це запобігає несанкціонованому доступу до чутливої фінансової інформації у разі витоку даних. Так само шифрування може захистити електронні листи, забезпечуючи конфіденційність особистих розмов, навіть коли вони надсилаються через незахищені мережі.

У бізнес-сфері компанії часто мають справу з чутливими даними, включаючи інформацію про клієнтів, власні дослідницькі дані та стратегічні плани, які необхідно захищати для підтримки конкурентних переваг та дотримання правил конфіденційності. Наприклад, компанія може використовувати додаток для шифрування файлів перед їх передачею віддаленим членам команди через інтернет. Це забезпечує безпеку даних під час передачі і дає доступ до них тільки членам команди, які мають ключ для розшифрування. Крім того, такий спосіб комунікації, може допомогти компаніям відповідати законам про захист даних, як-от GDPR, який вимагає захисту особистих даних.

Медична галузь також значною мірою користується шифруванням. Медичні записи містять дуже приватну інформацію, яку необхідно тримати в таємниці. Якщо ця інформація буде розкрита, це може порушити конфіденційність та бути використаною неналежним чином.

У державному управлінні теж дуже важливо захищати конфіденційну інформацію. Державні установи можуть використовувати шифрування для

захисту документів, які стосуються національної безпеки, внутрішніх обговорень та іншої конфіденційної інформації. Це допомагає запобігти витокам, які могли б нашкодити національній безпеці чи міжнародним відносинам.

Також можна згадати сектор журналістики. Журналісти часто мають справу з конфіденційними джерелами, які потрібно зберегти в таємниці. Шифруючи свої комунікації та дані, журналісти можуть забезпечити безпеку ідентичності своїх джерел та наданої інформації, що є критично важливим у регіонах, де свобода преси не гарантована.

3.2 Демонстрація кодування та розкодування тексту та файлів

3.2.1 Процес шифрування та дешифрування текстових даних

Процес шифрування тексту за допомогою алгоритму AES у програмі Python містить кілька простих кроків. Починаючи, користувач взаємодіє з графічним інтерфейсом програми, який організований на декілька секцій для зручності користування. Ось як відбувається процес шифрування крок за кроком:

- Відкриття вікна тексту: Користувач починає з натискання кнопки "Text", позначеної як "1" (рис. 3.1). Ця дія відкриває нове вікно, спеціально призначене для текстових операцій, забезпечуючи чистий і присвячений простір для роботи з текстовими даними.
- Вибір вкладки шифрування: У вікні тексту є кілька вкладок для різних операцій. Користувачеві потрібно вибрати вкладку "Encryption", позначену як "2" (рис. 3.1). Ця вкладка містить усі необхідні інструменти та поля для введення, необхідні для шифрування тексту.
- Введення тексту: У текстовому полі "Input Text", позначеному як "3" (рис. 3.1), користувач вводить текст, який він хоче зашифрувати. Для цього прикладу використовується текст "Some really important secret message".

Важливо, щоб текст був введений правильно, оскільки будь-яка помилка в написанні або пунктуації вплине на зашифрований вихідний текст.

- Встановлення паролю: Далі користувач вводить пароль у текстове поле "password", ідентифіковане як "4" (рис. 3.1). Цей пароль є ключовим, оскільки служить ключем для шифрування AES. У цьому випадку оберемо пароль "MySecretPassword01".
- Шифрування тексту: Після введення тексту та налаштування паролю користувач натискає кнопку "Encrypt", позначену як "5" (рис. 3.1).

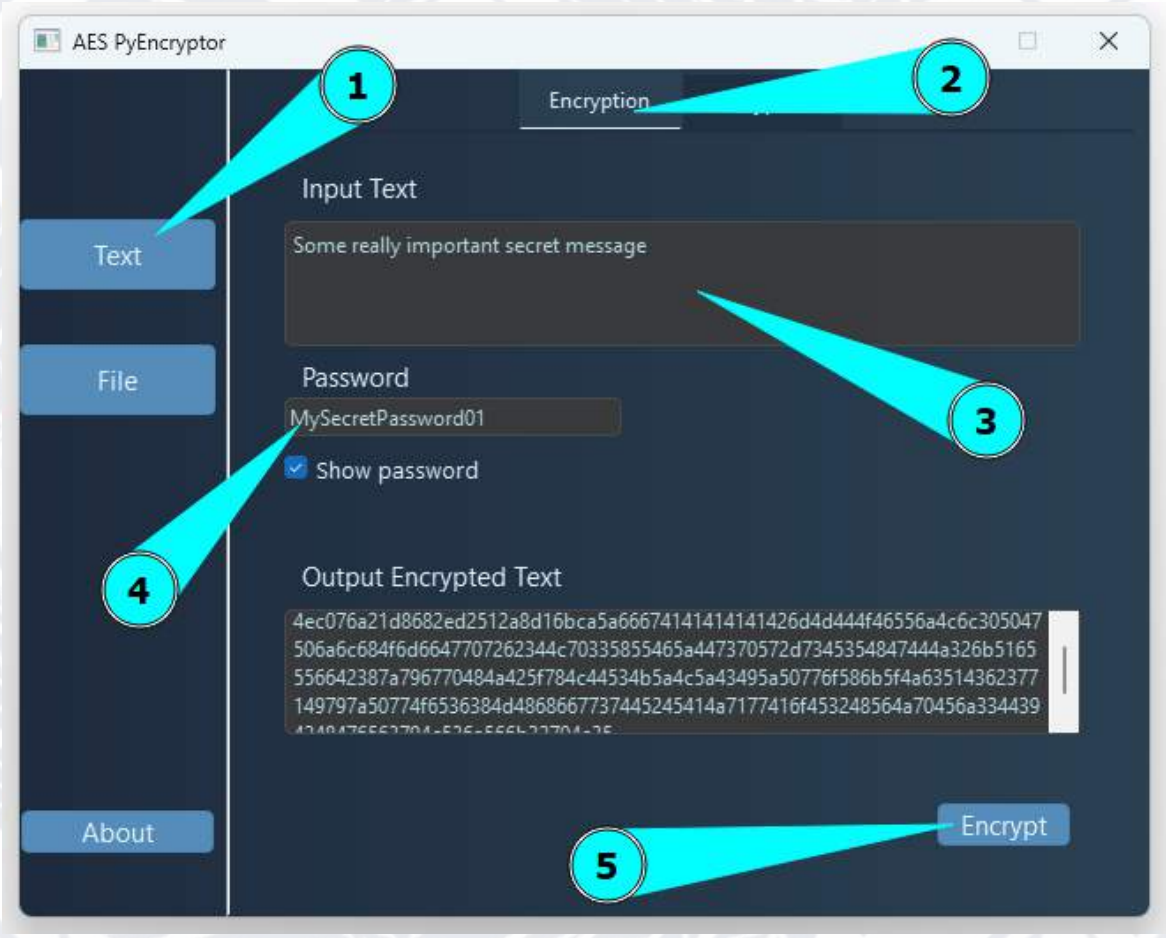


Рисунок 3.1 – Процес шифрування тексту

Результат процесу шифрування відображається у текстовому полі "Output Encrypted Text". Для введеного тексту зашифроване повідомлення з'являється у вигляді довгого рядка шістнадцяткових символів:

“4ec076a21d8682ed2512a8d16bca5a66674141414141426d4d444f46556a4c6c305047506a6c684f6d6647707262344c70335855465a447370572d7345354847444a326b51

65556642387a796770484a425f784c44534b5a4c5a43495a50776f586b5f4a63514362
377149797a50774f6536384d4868667737445245414a7177416f453248564a70456a3
34439424847656379 4c536a566b32704e35”

Продовжуючи процес шифрування, розшифровка тексту виконується за подібною послідовністю дій, але в зворотному порядку.

Для початку розшифровки раніше зашифрованого тексту, користувач натискає на вкладку "Decryption" позначеної як "1" (рис. 3.2).

У текстовому полі "Input Encrypted Text", користувач вставляє зашифрований текст, який був згенерований під час процесу шифрування, що починається з "4ec076a21d8682...".

У текстовому полі "Password", позначеному як "2" (рис. 3.2), користувач вводить пароль, який використовувався під час шифрування. Однак у цьому випадку розглянемо ситуацію, коли користувач помилково вводить "MySecretPassword1", пропускаючи останню цифру '0', яка була частиною оригінального паролю "MySecretPassword01".

Після введення зашифрованого тексту та неправильного паролю, користувач натискає кнопку "Decrypt", позначену як "3" (рис. 3.2).

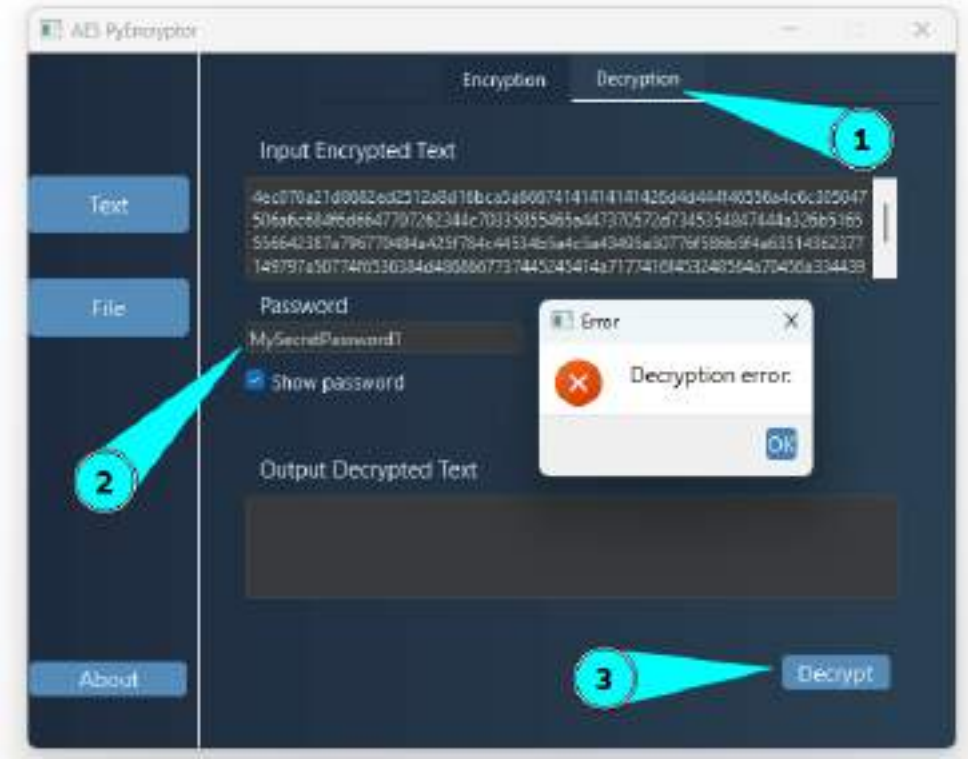


Рисунок 3.2 – Вікно не успішного розшифрування тексту

Оскільки введений пароль не співпадає з оригінальним паролем шифрування через відсутність символу, процес розшифровки не вдається. В результаті з'являється вікно з помилкою (рис. 3.2), яке відображає повідомлення про помилку розшифровки. Ця помилка підкреслює важливість точності при введенні паролю, оскільки навіть один неправильний або відсутній символ перешкоджає успішній розшифровці.

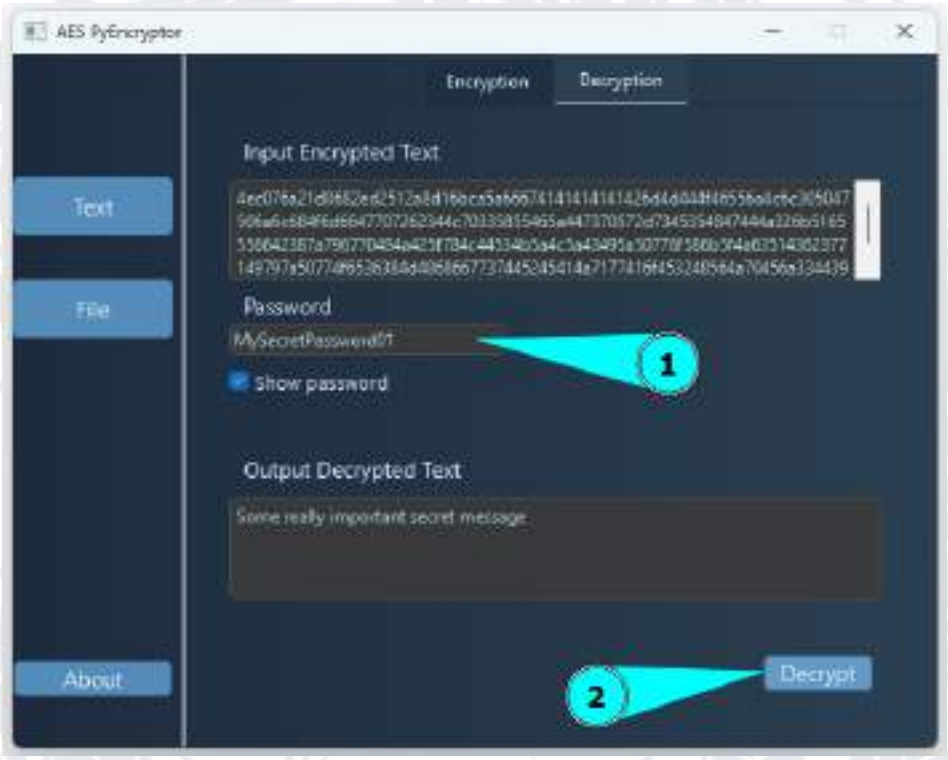


Рисунок 3.3 – Результат успішного розшифрування тексту

Крім правильного введення пароля, важливо ввести зашифрований текст точно так, як він з'являється. Будь-яка зміна, навіть одного символу, призведе до помилки. Повернувши оригінальний пароль – отримаємо наш початковий текст (рис. 3.3).

3.2.2 Процес шифрування та дешифрування файлів

Щоб розпочати процес шифрування, потрібно натиснути на кнопку “File” аби перейти на необхідну вкладку, після чого натиснути кнопку “OpenFile” у

програмному інтерфейсі, яка позначена як крок “1” (рис. 3.4). Ця дія спричиняє відкриття спливаючого вікна, яке дозволяє переглядати та вибирати файли, над якими необхідно виконати операцію (рис. 3.5). Оберемо для прикладу зображення від назвою “MySecretPhoto.jpg” та натиснемо “Open”.

Рисунок 3.4 – Вікно для шифрування файлів

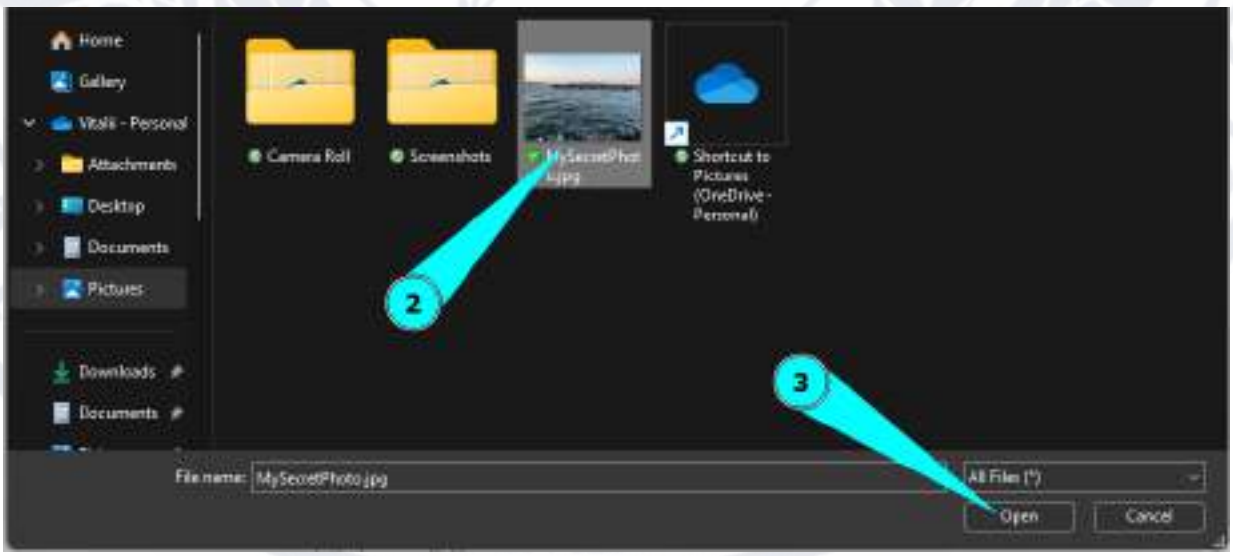


Рисунок 3.5 – Вікно перегляду файлів операційної системи

У текстовому полі "New File Name", позначеному як "4" (рис. 3.4), введемо назву для нового зашифрованого файлу. Для прикладу це буде “EncryptedPhoto.jpg”.

Наступний крок – це вибір операції "Encryption", або "Decryption" на радіо-кнопках, позначених як "5" (рис. 3.4). Наразі виберемо "Encryption".

Обов’язково потрібно ввести міцний пароль у поле “Password”, позначене як "6" (рис. 3.5). Після чого натискаємо кнопку "Encryption”.

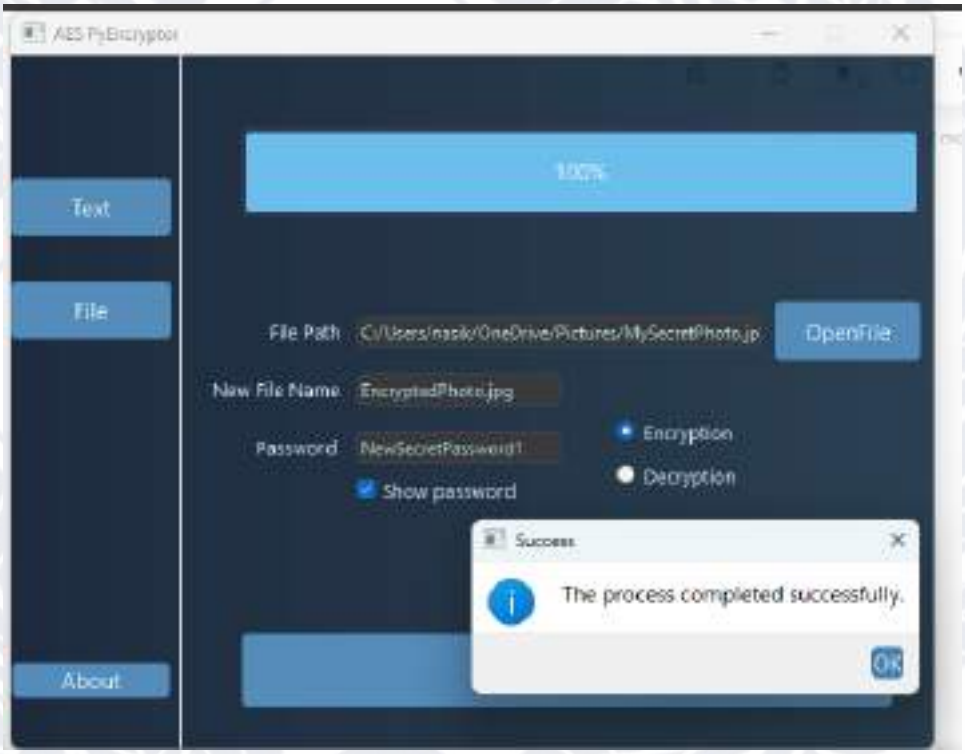


Рисунок 3.6 – Вікно успішного шифрування файлу

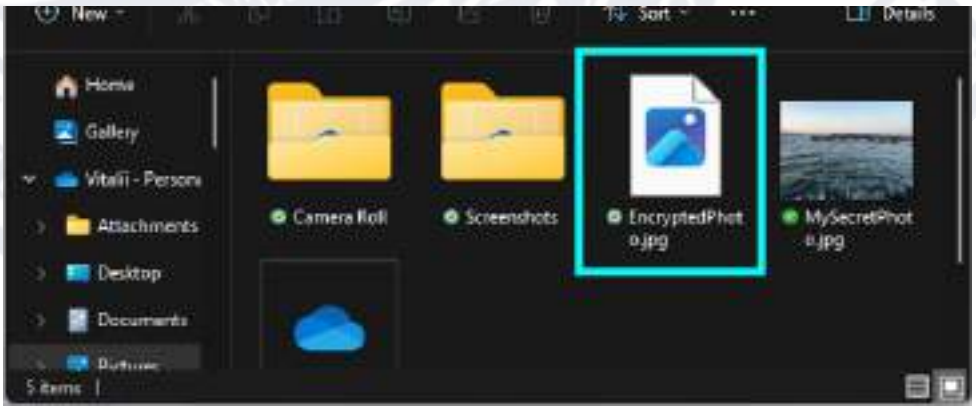


Рисунок 3.7 – Вікно файлової системи із шифрованим файлом

Після шифрування перейдемо до провідника файлів, щоб перевірити результати. І як бачимо на рисунку 3.7, у нас з’явився файл “EncryptedPhoto.jpg”. Спроба відкрити цей файл призведе до повідомлення про помилку від операційної системи, яка стверджує, що не може розпізнати або працювати з форматом файлу. Ця помилка підтверджує, що вміст файлу тепер захищений і недоступний без розшифрування. (рис. 3.8).

Рисунок 3.8 – Сповіщення помилки при натисканні на зображення

Тепер перейдемо до процесу розшифрування. Щоб отримати доступ до зашифрованих даних, виконаємо все теж саме, тільки в зворотному порядку:

- Оберемо зашифрований файл "EncryptedPhoto.jpg".
- Введемо "DecryptedPhoto.jpg" як нову назву файлу в інтерфейсі розшифрування.
- Оберемо радіо-кнопку "Decryption".
- Введемо правильний пароль, який використовувався під час шифрування.
- Натиснемо кнопку "Decryption".

Після розшифрування ми отримаємо повідомлення про успішну операцію (рис. 3.9). Як бачимо на рисунку 3.10 новий розшифрований файл з’явився в провіднику. При натисканні на файл операційна система не видає ніяких помилок і відкриває ідентичне зображення, яке на початку було обрано в якості прикладу.

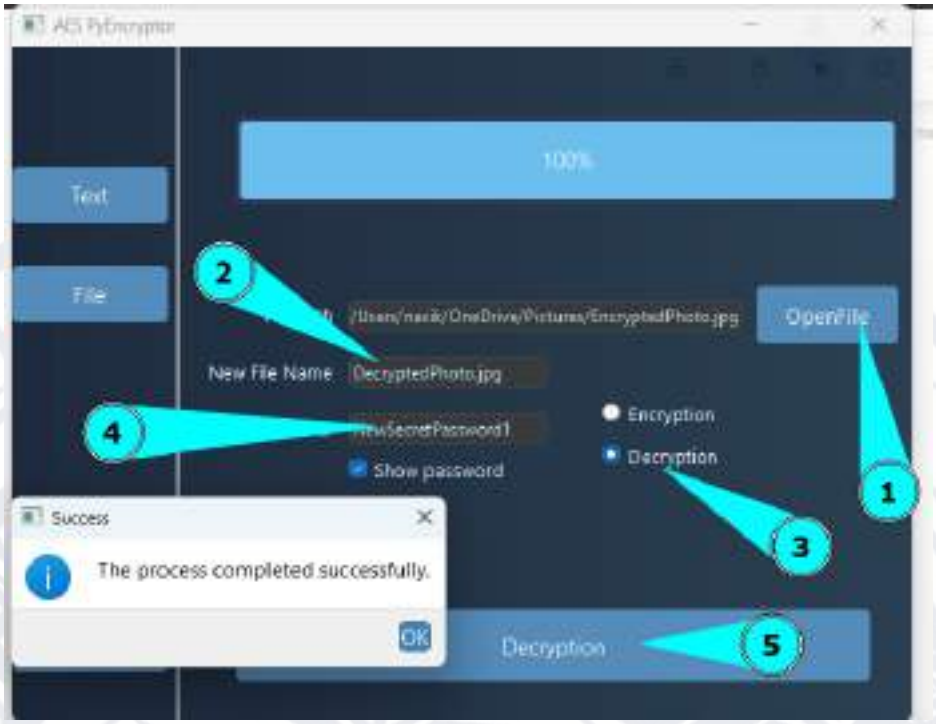


Рисунок 3.9 – Сповіщення про успішне виконання розшифрування зображення

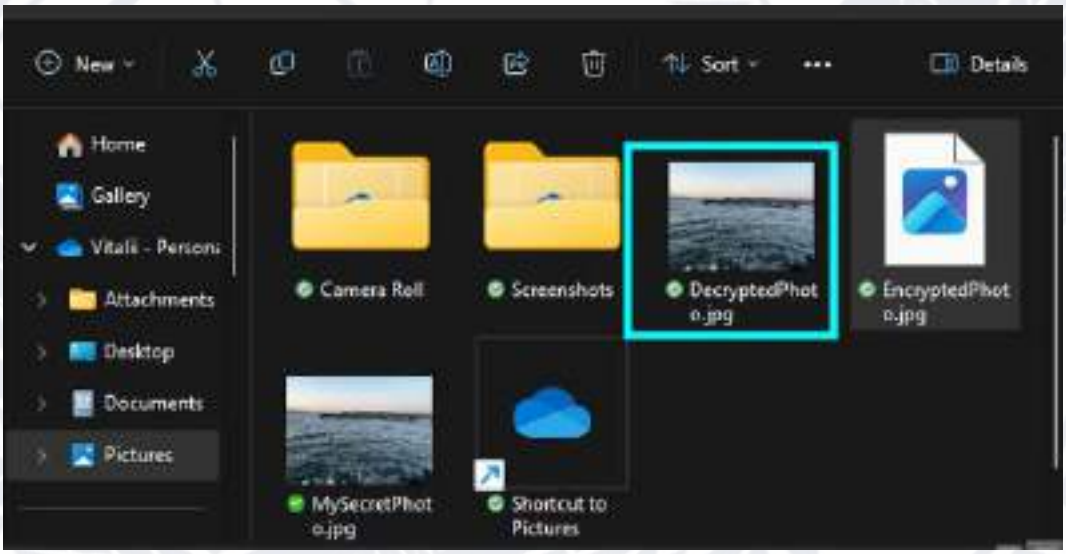


Рисунок 3.10 – Вікно файлової системи із розшифрованим файлом

Щоб перевірити цілісність розшифрованого файлу, порівняємо значення хешів оригінального і розшифрованого файлів за допомогою функції “Get-FileHash” у PowerShell (рис. 3.11). Відповідність хешів підтверджує, що процес розшифрування відновив оригінальний файл ідеально, без втрати даних або пошкоджень.

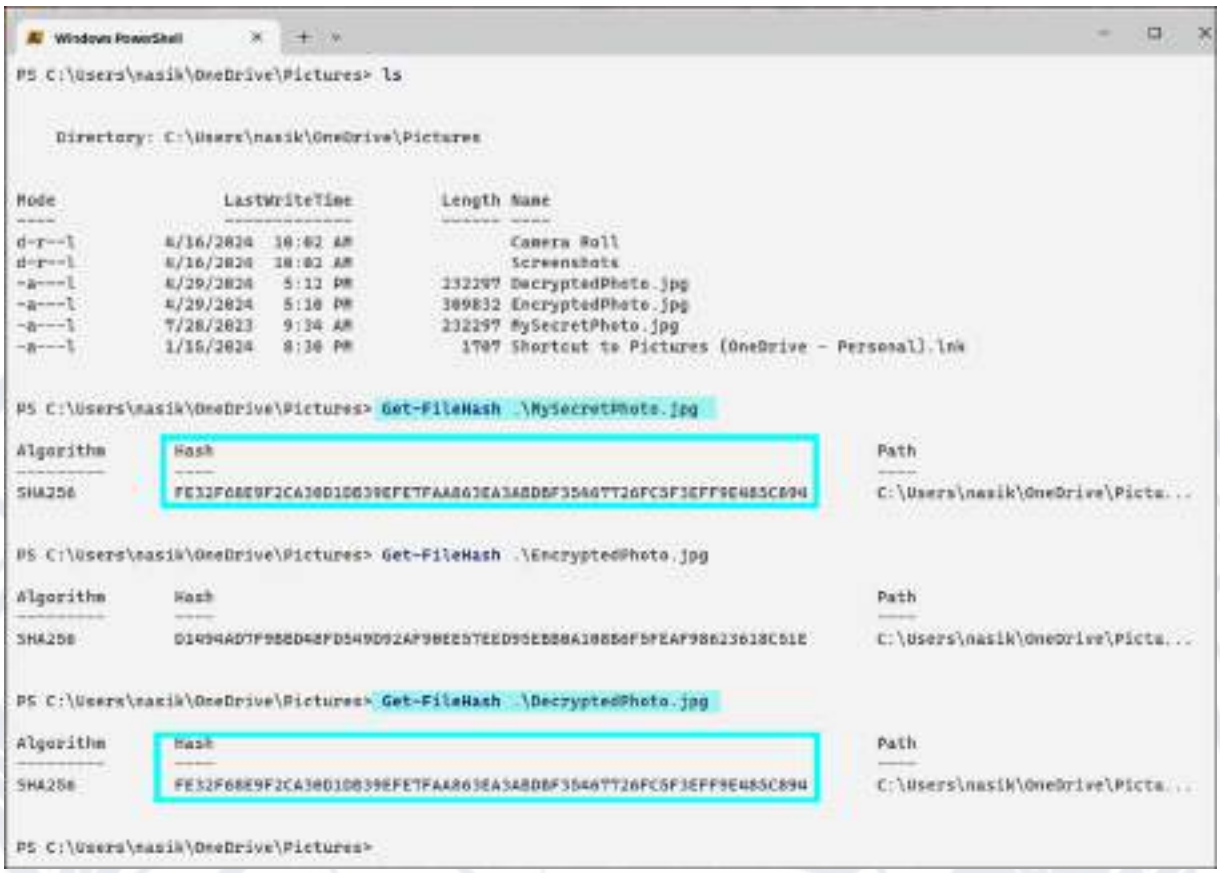


Рисунок 3.11 – Вікно PowerShell із результатами виконання “Get-FileHash”

В третьому розділі були розглянуті приклади шифрування та дешифрування тексту, а також робота із зашифрованими файлами. Було показано, як перевіряти цілісність файлів після процесу модифікації та як потрібно взаємодіяти із графічним інтерфейсом додатку.

ВИСНОВКИ

Результатом дослідження цієї роботи став додаток для захисту інформації у форматі тексту та файлів. Завдяки використанню останніх криптографічних стандартів, реалізованих у проекті та можливості перегляду відкритого коду, можна без сумнівів стверджувати, що додаток може бути використаний без будь-яких занепокоєнь щодо безпеки приватних даних, які шифруються.

У рамках роботи було досліджено основну проблему користувачів, які втрачають приватні дані внаслідок кібератак, а також розглянуто способи їх захисту. Необхідність та актуальність цієї роботи надзвичайно висока через зростання загроз, пов’язаних із розвитком мережі Інтернет та через нестачу аналогів із відкритим кодом, які гарантовано не збирають особисті дані користувачів.

Було також розглянуто існуючі інструменти, за допомогою яких можна швидко реалізувати продукт та задовольнити максимум вимог. Після вибору всіх технологій та платформ для створення, було проаналізовано процес побудови та тестування безпосередньо тестового продукту.

Підсумовуючи, можна зазначити, що завдання цієї кваліфікаційної роботи були виконані у повному обсязі.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. KR. Laboratories. Безпека як стан захищеності і природна потреба людства. Вступ до основ кібербезпеки. 2023 [Електронний ресурс] – Режим доступу до ресурсу: kr-labs.com.ua/blog/introduction-to-cybersecurity (дата звернення 08.03.2024)
2. Мазур Ю. О., Зелінська О. В. Особливості захисту сучасної інфосфери в умовах стороннього кібернетичного впливу. Прикладні аспекти сучасних міждисциплінарних досліджень. 2021, С. 102-103
3. Rommel Joven, Ng Choon Kiat. The Spies Who Loved You: Infected USB Drives to Steal Secrets [Електронний ресурс] – Режим доступу до ресурсу: www.mandiant.com/resources/blog/infected-usb-steal-secrets (дата звернення 30.03.2024)
4. Rob Sobers. 84 Must-Know Data Breach Statistics. 2023 [Електронний ресурс] – Режим доступу до ресурсу: www.varonis.com/blog/data-breach-statistics (дата звернення 30.03.2024)
5. SentinelOne. What Is A Malware File Signature (And How Does It Work)? 2021 [Електронний ресурс] – Режим доступу до ресурсу: www.sentinelone.com/blog/what-is-a-malware-file-signature-and-how-does-it-work (дата звернення 30.03.2024)
6. Jessica Davis. Hackers Successfully Exploiting Older, Unpatched Microsoft Vulnerabilities. 2021 [Електронний ресурс] – Режим доступу до ресурсу: www.healthitsecurity.com/news/hackers-successfully-exploiting-older-unpatched-microsoft-vulnerabilities (дата звернення 07.04.2024)
7. Tresorit Team. The history of encryption: the roots of modern-day cyber-security. 2021 [Електронний ресурс] – Режим доступу до ресурсу: tresorit.com/blog/the-history-of-encryption-the-roots-of-modern-day-cyber-security (дата звернення 07.04.2024)

8. Nicolas Poggi. Types of Encryption: Symmetric or Asymmetric? RSA or AES? 2021 [Електронний ресурс] – Режим доступу до ресурсу: preyproject.com/blog/types-of-encryption-symmetric-or-asymmetric-rsa-or-aes (дата звернення 07.04.2024)
9. Telegram. API End-to-End Encryption, Secret Chats. [Електронний ресурс] – Режим доступу до ресурсу: core.telegram.org/api/end-to-end (дата звернення 07.04.2024)
10. WhatsApp. API About end-to-end encryption. [Електронний ресурс] – Режим доступу до ресурсу: faq.whatsapp.com/820124435853543 (дата звернення 07.04.2024)
11. Geeksforgeeks. Difference between SSH and SSL. 2023 [Електронний ресурс] – Режим доступу до ресурсу: www.geeksforgeeks.org/difference-between-ssh-and-ssl (дата звернення 07.04.2024)
12. Javapoint. Difference Between DES (Data Encryption Standard) and AES (Advanced Encryption Standard). [Електронний ресурс] – Режим доступу до ресурсу: www.javatpoint.com/des-vs-aes (дата звернення 14.04.2024)
13. Geeksforgeeks. Advanced Encryption Standard (AES). 2023 [Електронний ресурс] – Режим доступу до ресурсу: www.geeksforgeeks.org/advanced-encryption-standard-aes (дата звернення 14.04.2024)
14. William Buchanan. Cryptography. MLA 9th Edition (Modern Language Assoc.) River Publishers, 2017. 63 с.
15. Geeksforgeeks. “12 Reasons Why You Should Learn Python (2024)” [Електронний ресурс] – Режим доступу до ресурсу: www.geeksforgeeks.org/reasons-why-you-should-learn-python/ (дата звернення 04.05.2024)
16. Python Software Foundation. tkinter — Python interface to Tcl/Tk. [Електронний ресурс] – Режим доступу до ресурсу: docs.python.org/3/library/tkinter.html (дата звернення 04.05.2024)

17. Riverbank Computing. PyQt. [Електронний ресурс] – Режим доступу до ресурсу: riverbankcomputing.com/software/pyqt/intro (дата звернення 05.05.2024)
18. The Qt Company. Qt for Python. [Електронний ресурс] – Режим доступу до ресурсу: doc.qt.io/qtforpython-6/ (дата звернення 05.05.2024)
19. Individual Contributors. Fernet (symmetric encryption). [Електронний ресурс] – Режим доступу до ресурсу: cryptography.io/en/latest/fernet/ (дата звернення 06.05.2024)
20. GitHub. Get started with GitHub. [Електронний ресурс] – Режим доступу до ресурсу: docs.github.com/en/get-started (дата звернення 06.05.2024)
21. GitLab. Get started with GitLab. [Електронний ресурс] – Режим доступу до ресурсу: docs.gitlab.com/ (дата звернення 06.05.2024)
22. Git, Documentation. [Електронний ресурс] – Режим доступу до ресурсу: git-scm.com/docs (дата звернення 08.05.2024)
23. GNU. General Public License. [Електронний ресурс] – Режим доступу до ресурсу: www.gnu.org/licenses/old-licenses/gpl-2.0.en.html (дата звернення 08.05.2024)
24. Баркалов О. О., Бабаков Р. М. Дослідження продуктивності кросплатформних бібліотек для побудови графічного користувацького інтерфейсу. Прикладні аспекти сучасних міждисциплінарних досліджень. 2022, С. 78-81
25. І.Л.Бородкіна, Г.О.Бородкін. Web-технології та Web-дизайн : застосування мови HTML для створення електронних ресурсів. Видавництво:Київ , 2020. 212 с.
26. Кордзая Н.Р., Основи інтернет-маркетингу. Частина 2. Видавництво: Херсон , 2018. 158 с.
27. Брэд Уильямс, Джон Джеймс Джейкоби, Джастин Тадлок., Книга Professional WordPress Plugin Development. Видавництво: Wiley. John Wiley & Sons, LTD, 2020. 480 с.
28. Роберт Мартин., Чистий кодер., Видавництво: Фабула. 2023 . 256 с.

29. Робін Ніксон. Створюємо динамічні веб-сайти за допомогою PHP, MySQL, JavaScript, CSS і HTML5., Видавництво: Кив. 2019 . 816 с
30. Роберт Сесил Мартин. Книга Чиста архітектура., Видавництво: Фабула, 2019. 368 с.
31. С. Ботрос, Д. Тинли. MySQL по максимуму. 4-те видання. Видавництво: Print2print. 2023. 432 с.
32. Алексей Васильев. Книга Програмування мовою PHP., Видавництво: Ліра-К. 2022 . 368 с
33. Патрик Дебуа, Джон Уиллис, Джин Ким, Джез Хамбл. Книга DevOps. Посібник., Видавництво: Фабула. 2023. 384 с
34. Мартін Р. Чиста архітектура. Видавництво: Фабула. 2020 . 416 с
35. Д. Олтаржевский, Е. Загорулько., Электронная книга Корпоративні комунікації. Свіжий погляд. Видавництво: Арт Економі. 2023 . 360 с.
36. Jesse Liberty. "Programming C# 8.0: Build Windows, Web, and Desktop Applications". O'Reilly Media, 2020. 878 p.
37. Bill Wagner. "C# 9.0 in a Nutshell: The Definitive Reference". O'Reilly Media, 2021. 1068 p.
38. John Sharp. "Microsoft Visual C# Step by Step". Microsoft Press, 2018. 816 p.
39. Jason De Oliveira, Michel Bruchet. "Building Single Page App With ASP.NET Core and Angular". Apress, 2019. 544 p.
40. Adam Freeman. "Pro ASP.NET Core Identity". Apress, 2019. 723 p.

ДОДАТКИ