МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ
«КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КІБЕРБЕЗПЕКИ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри кібербезпеки

_____Анна ІЛЬЄНКО
"_____"_____ 20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ "МАГІСТР"

Тема: Програмний застосунок для шифрування ресурсів

Виконавець: Андрій СМАЛЬ

Керівник: к.т.н., доцент Анна ІЛЬЄНКО

Нормоконтролер: к.т.н., доцент Андрій ПЕТРЕНКО

Київ 2024

ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ
«КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Факультет комп’ютерних наук та технологій
Кафедра кібербезпеки
Освітній ступінь магістр
Спеціальність 125 «Кібербезпека та захист інформації»
Освітньо-професійна програма «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки

_____Анна ІЛЬЄНКО
«30»___08_____2024 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Смаля Андрій Олександровича

1. Тема кваліфікаційної роботи: Програмний застосунок для шифрування ресурсів
затверджена наказом ректора від 30.08.2024 р. №1695/ст.
2. Термін виконання роботи: з 30.08.2024 по 15.12.2024
3. Вихідні дані до роботи: проаналізувати сучасний стан захисту інформації; на основі аналізу створити програмний застосунок для шифрування ресурсів; провести порівняння цього застосунку з існуючими аналогами
4. Зміст пояснювальної записки: аналіз сучасного стану захисту інформації; опис обраних методів забезпечення безпеки, практична реалізація програмного застосунку для шифрування ресурсів
5. Перелік обов'язкового графічного (ілюстративного) матеріалу: презентація

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Уточнення постановки завдання	30.08.2024	*Виконано*
2.	Збір та аналіз літературних джерел	30.08.2024 – 07.09.2024	*Виконано*
3.	Аналізування сучасного стану захисту інформації	09.09.2024 – 25.09.2024	*Виконано*
4.	Розробка алгоритму застосунку	25.09.2024 – 15.10.2024	*Виконано*
5.	Програмна реалізація розробленого алгоритму	16.10.2024 – 27.10.2024	*Виконано*
6.	Проведення тестування створеного програмного застосунку	27.10.2024 – 01.11.2024	*Виконано*
7.	Оформлення презентації та пояснювальної записки	02.11.2024 – 18.11.2024	*Виконано*

7. Дата видачі завдання: «30» __08__ 2024 р.

Керівник кваліфікаційної роботи: _____ Анна ІЛЬЄНКО
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання: _____ Андрій СМАЛЬ
(підпис здобувача вищої освіти) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмний застосунок для шифрування ресурсів»: 103 с., 46 рис., 9 табл., 41 літературне джерело, 1 додаток.

Об’єкт дослідження: процес захисту ресурсів шляхом шифрування з використанням системи розпізнавання обличчя.

Предмет дослідження: методи захисту ресурсів, що використовують біометричні дані та криптографію для підвищення надійності захисту.

Мета кваліфікаційної роботи: розробка програмного застосунку системи шифрування даних з використанням біометричних даних обличчя.

Методи дослідження: сучасні методи побудови застосунків для шифрування даних, управління паролями, роботи з біометричними даними.

Практична цінність: програмний застосунок для захисту ресурсів з використанням мови програмування Python, за рахунок інтеграції системи розпізнавання облич дозволяє забезпечити високий рівень безпеки даних не поступаючись при цьому зручністю користування.

Наукова новизна: розроблено авторський програмний застосунок шифрування інформації, з використання можливості забезпечення захисту головного ключа з використанням паролю та біометричних характеристик користувача, що дозволяє забезпечити цілісність та конфіденційність ресурсів. Застосунок можна легко інтегрувати в уже використовувані інформаційні системи, що значно розширює його практичне застосування та спрощує впровадження у великих організаціях. Також, на відміну від відомих рішень, забезпечує більший рівень безпеки та гнучкості за рахунок використання рольової моделі обмеження доступу.

Результати кваліфікаційної роботи рекомендується використовувати для інтеграції в існуючі інформаційні системи, щоб значно зменшити ризик несанкціонованого доступу до ресурсів користувачів.

ЗМІСТ

ВСТУП

Актуальність теми. На тлі стрімкого розвитку цифрових технологій і глобальної цифровізації суспільства питання забезпечення інформаційної безпеки стає одним із найважливіших викликів сучасності. Зростання обсягів персональних і корпоративних даних, які зберігаються в електронному вигляді, ускладнює контроль доступу до цих даних і збільшує ймовірність їх втрати або компрометації. Кіберзагрози стають все більш складними й витонченими, що потребує впровадження інноваційних методів захисту.

Традиційні підходи до шифрування зазвичай обмежені лише паролями та/або апаратними рішеннями. Хоча ці методи широко використовуються, вони мають значні обмеження. Паролі можуть бути такими, що легко підібрати, або викрасти. Апаратні рішення, такі як смарт-карти або апаратні ключі, хоч і забезпечують вищий рівень безпеки, але можуть бути втрачені, викрадені або пошкоджені, та не дуже зручні в використанні для користувачів.

Водночас, розвиток біометричних технологій відкриває нові можливості для підвищення рівня безпеки. Біометричні дані, зокрема розпізнавання обличчя, мають ряд переваг, які роблять їх перспективним рішенням для забезпечення конфіденційності інформації. Унікальність біометричних характеристик кожної людини грає в цьому основну роль. Розпізнавання обличчя є одним із найбільш зручних і швидких способів біометричної ідентифікації, оскільки не потребує контакту з обладнанням і може бути інтегроване в майже будь-який цифровий пристрій, що обладнаний камерою.

Актуальність цієї роботи обґрунтована поєднанням звичайних алгоритмів шифрування та системи розпізнавання обличчя в один застосунок, що підвищує безпеку, не поступаючись при цьому зручністю користування. Інтеграція цих технологій у єдиний застосунок дозволяє створити рішення, яке одночасно задовольняє потреби у високому рівні безпеки та зручності використання.

Мета і завдання виконання кваліфікаційної роботи. Мета проєкту є розробка системи шифрування даних з використанням біометричних даних обличчя. Виходячи з мети, завданням даної дипломної роботи є:

- дослідження сучасних методів шифрування та розпізнавання обличчя;
- розробка програмного застосунку для шифрування ресурсів з використанням біометрії обличчя;
- оцінка ефективності результатів та порівняння з аналогами.

Об’єкт і предмет дослідження.

Об'єкт дослідження: процес захисту ресурсів шляхом шифрування з використанням системи розпізнавання обличчя.

Предмет дослідження: методи захисту ресурсів, що використовують біометричні дані та криптографію для підвищення надійності захисту.

Методи дослідження.

Для досягнення поставленої мети проведені дослідження основані на сучасних методах побудови застосунків для шифрування даних, управління паролями, роботи з біометричними даними.

Наукова новизна отриманих результатів.

Новизна роботи полягає у запропонованому авторському програмному застосунку для забезпечення безпеки даних, що має за основу сучасний симетричний алгоритм шифрування тісно пов’язаний з системою роботи з біометричними даними. Розроблено новий підхід до шифрування, де головний ключ одночасно захищається двома різними механізмами: паролем і унікальними біометричними даними обличчя. На відміну від відомих рішень, запропонована система забезпечує більший рівень безпеки та гнучкості за рахунок використання рольової моделі обмеження доступу.

Практичне значення отриманих результатів.

Розроблений програмний застосунок для захисту ресурсів з використанням мови програмування Python, за рахунок інтеграції системи

розпізнавання облич дозволяє забезпечити безпеку даних не поступаючись при цьому зручністю користування.

Апробація отриманих результатів. Отримані в процесі виконання кваліфікаційної роботи результати були апробовані на десятій міжнародній науково-практичній конференції «European congress of scientific achievements». Посилання: Смаль А. О., Ільєнко А. В. Технології шифрування ресурсів // European congress of scientific achievements. Proceedings of the 10th International scientific and practical conference. Barca Academy Publishing. Barcelona, Spain. 2024. Pp. 132-136. URL: https://sci-conf.com.ua/x-mizhnarodna-naukovo-praktichna-konferentsiya-european-congress-of-scientific-achievements-7-9-10-2024-barselona-ispaniya-arhiv/

РОЗДІЛ 1
АНАЛІЗ СУЧАСНОГО СТАНУ ЗАХИСТУ ІНФОРМАЦІЇ

1.1 Стан та проблеми забезпечення безпеки інформації в інформаційних мереж

З темпом сучасного розвитку технологій і збільшенням обсягу оброблюваних даних, зростає необхідність захищати інформацію від несанкціонованого доступу. З інтеграцією інтернету в усі аспекти суспільства інформаційні мережі набули величезної ваги, забезпечуючи обмін і доступ до інформації у масштабах, які раніше здавалися неможливими. Водночас треба розуміти, що з цими можливостями виникають і нові загрози, які ставлять під сумнів безпеку даних, що передаються та обробляються в таких мережах.

Загроза – будь-які обставини або події, що можуть бути причиною порушення політики безпеки інформації та/або завдавати збитків автоматизованій системі. Найбільш поширені загрози:

Шкідливе програмне забезпечення. Вимагачі, трояни, хробаки, віруси та інші, часто комбіновані типи шкідливих програм, що здатні проникати в інформаційні мережі, пошкоджувати, змінювати та викрадати інформацію, при цьому не тільки порушуючи нормальну роботу систем, але й створюючи лазівки для інших атак.

- Вимагачі. Цей вид шкідливого програмного забезпечення використовується кіберзлочинцями для вимагання коштів у жертв. Зазвичай, програма-вимагач блокує екран пристрою жертви та/або шифрує дані на диску, відображаючи вимогу про викуп із реквізитами платежу, та таймером до видалення даних.
- Трояни. Даний тип шкідливого програмного забезпечення є найпоширенішим, його суть полягає в маскуванні під нормальне

програмне забезпечення. Трояни не вміють поширюватися самостійно, розповсюджується самими користувачами.

- Віруси. Цей тип зловмисного програмного забезпечення, який при запуску розмножується, роблячи копії, змінюючи інші комп’ютерні програми вставляючи в ці програми власний код. Складаються з трьох частин: механізм зараження, для знаходження та зараження нових файлів; корисне навантаження, яке є шкідливим кодом для виконання; тригер, який визначає, коли активувати корисне навантаження (умовою може бути конкретна дата та час, відкриття певного файлу, встановлення програми тощо).
- Хробаки. Комп’ютерна програма, що підпадає під даний тип зловмисного програмного забезпечення, також, як і вірус, робить копії самої себе, але окрім цього може поширюватися на інші комп’ютери за допомогою комп’ютерної мережі. Вона сканує мережу на наявність інших комп’ютерів та намагається заразити їх, зазвичай робить це з мінімальною шкодою для хоста(-ів), лише навантажуючи мережу, коли віруси майже завжди пошкоджують або змінюють файли при розповсюдженні.

Соціальна інженерія. Форма атаки, яка спрямована на використання особливостей людської психіки. Через це надійність комп'ютерної системи є не вищою, ніж надійність її оператора. За допомогою обману працівників або користувачів зловмисники проникають в добре спроєктовані, самі захищені комп'ютерні системи, змушуючи їх розкрити конфіденційну інформацію або виконати небезпечні для безпеки системи дії. Самий популярним варіантом є фішинг, сьогодні основна складова фішингу – шахрайські вебресурси, що копіюють вебресурс цільової організації, наприклад, банкінгу.

DDoS-атаки (розподілена атака на відмову в обслуговуванні). Цей тип атаки спрямований на перевантаження ресурсів мережевої інфраструктури, що призводить до недоступності сервісів та систем. Суть атаки полягає в одночасному здійснюванні за допомогою безлічі хостів великої кількості певних

запитів. Організуються або добровільно великою кількістю людей, або з ботнетів (комп'ютерна мережа, що складається з деякої кількості хостів, із запущеними програмами, які приховано встановлюється на комп'ютери жертви, що дозволяє зловмисникові виконувати певні дії з використанням ресурсів інфікованих комп'ютерів).

Атака посередника – ситуація, коли зловмисник має можливість читати та/чи змінювати повідомлення, якими обмінюються користувачі, при цьому жоден з них не знає про його присутність в каналі.

Вразливості програмного забезпечення. Навмисні та випадкові помилки у програмних продуктах можуть створювати серйозні ризики для інформаційної безпеки, відкриваючи можливості для зловмисників використати ці вразливості для атак.

Стан забезпечення безпеки інформаційних мереж

У зв'язку з поширенням кіберзагроз, організації по всьому світу активно впроваджують системи захисту інформації, такі як міжмережеві екрани, системи виявлення вторгнень, системи шифрування даних та інші засоби захисту, проте, попри значний прогрес у розробці таких систем, проблеми забезпечення інформаційної безпеки залишаються актуальними.

Згідно з дослідженнями у галузі кібербезпеки, більшість інцидентів інформаційної безпеки стаються через людський фактор, недостатнє налаштування безпеки, застаріле програмне забезпечення та відсутність належного моніторингу активності в мережах. Попри розвиток технологій захисту, атаки постійно вдосконалюються та стають більш складними [1].

Існують декілька важливих аспектів, що характеризують стан інформаційної безпеки в сучасних інформаційних мережах:

- Низька обізнаність користувачів. Користувачі часто не мають достатнього рівня знань про основні методи захисту, через що можуть стати мішенню для фішингових атак або інших форм соціальної інженерії.

- Інтеграція нових технологій без достатньої перевірки. Впровадження нових технологій, таких як Інтернет речей (IoT), хмарні обчислення, створює нові можливості для зловмисників через уразливості в цих системах.
- Недостатнє фінансування заходів з безпеки. Деякі організації, особливо малі та середні підприємства, не мають достатніх фінансових ресурсів для впровадження сучасних систем захисту, що робить їх більш вразливими до атак.

Якщо підсумувати цей підрозділ, то можна сказати, що сучасний стан забезпечення безпеки інформаційних мереж залишається складним і вимагає постійного вдосконалення засобів захисту, а також підвищення рівня обізнаності та компетенцій користувачів.

1.2. Тенденції розвитку технологій шифрування для забезпечення інформаційної безпеки

Шифрування даних використовується для захисту конфіденційності даних шляхом перетворення їх у незрозумілу форму, яка може бути розшифрована лише з використанням ключа. Тобто, наприклад, якщо звичайний користувач якимось чином зможе отримати доступ до зашифрованих файлів адміністратора, то без відповідних ключів він не зможе прочитати вміст цих файлів.

Для чого воно може використовуватися. Враховуючи все більшу цифровізацію, все більше чутливих даних зберігаються в цифровому вигляді, тому шифрування для захисту конфіденційної інформації є критично важливим в умовах зростаючого числа кіберзагроз і нормативних вимог. Наприклад, персональні дані користувачів, фінансові транзакції, дані про здоров'я – усе це повинно бути зашифровано, аби зловмисники не змогли використати ці дані в разі їх витоку. Важливу роль в цьому грають нормативні акти, як-от GDPR

(регламент в межах законодавства Європейського Союзу щодо захисту персональних даних усіх осіб у межах Європейського Союзу та Європейської економічної зони), відповідно до якого, необхідно оцінювати властиві обробці інформації ризики, та вжити відповідних заходів для зменшення цих ризиків, наприклад шифрування та псевдонімізація. Також GDPR вимагає, щоб додаткова інформація (ключ дешифрування підпадає під це) зберігалася окремо від псевдонімізованих даних. Комунікація з користувачами має бути захищена: особиста інформація або конфіденційні дані користувача мають бути зашифровані, щоб зберегти конфіденційність; навіть доступ до загальнодоступних ресурсів має бути захищений за допомогою шифрування, щоб запобігти можливості профілювання користувачів, аналізу їх поведінки, вилучення ідентифікаторів, які можуть бути використані для подальшого відстеження [2, 3].

Окремої уваги заслуговує банківська сфера. З популяризацією та активним використанням платіжних карток з'явилися й нові виклики, компанії, щоб на них відповісти, впроваджували різні програми захисту, але були проблеми сумісності між існуючими стандартами. Саме тому в 2004 був розроблений та впроваджений по всьому світу стандарт PCI DSS. Стандарт був розроблений Радою зі стандартів безпеки індустрії платіжних карток, яка була заснована міжнародними платіжними системами, такими як Visa, MasterCard тощо. Поміж з ряду вимог, які цей стандарт включає він вимагає й шифрування даних власника картки при передачі у відкритих публічних мережах. Крім того, Стандарт передбачає перевірку відповідності, тобто оцінку та підтвердження того, що засоби контролю та процедури безпеки було впроваджено відповідно до PCI DSS. Перевірка відбувається через щорічну оцінку, або зовнішньою організацією, або шляхом самооцінки. Visa та Mastercard вимагають від постачальників послуг проходження даної перевірки, але Visa також пропонує альтернативу, яка дозволяє кваліфікованим продавцям припинити щорічну

перевірку. Для цього вони повинні вжити альтернативні заходи проти шахрайства, наприклад шифрування «Point-to-Point Encryption» [4].

Основні види шифрування

Існують два основні типи шифрування: симетричне та асиметричне. Симетричне шифрування використовує один і той самий ключ для шифрування та дешифрування даних. До переваг належать хороша теоретична вивченість, у тому числі обґрунтування криптостійкості, висока швидкість та ефективність, що особливо помітно при роботі з великими обсягами даних. Основною ж проблемою симетричного шифрування є необхідність безпечного обміну ключами між учасниками зв'язку [5].

Найпоширенішим алгоритмом симетричного шифрування є AES (Advanced Encryption Standard), який є варіантом шифру Rijndael, розроблений бельгійськими криптографами Джоаном Даеменом і Вінсентом Рейменом, був одним із конкуруючих проєктів на заміну DES (йому заміну шукали через його коротку довжину ключа в 56 біт, що зробила його занадто вразливим до атаки перебором). Він виграв 5-річний публічний конкурс на отримання статусу AES (Advanced Encryption Standard).

Прийнятий NIST у 2001 році, AES має фіксований розмір блоку 128 біт і розмір ключа 128, 192 або 256 біт, тоді як Rijndael може вказати розміри блоків і ключів у будь-якій кількості, кратній 32 бітам, з мінімумом 128. біти. Розмір блоку має максимум 256 біт, але розмір ключа не має теоретичного максимуму. AES працює на основі матриці порядку байтів 4×4 із великими стовпцями (версії Rijndael із більшим розміром блоку мають додаткові стовпці) [6]. В асиметричному шифруванні, на відміну від симетричного шифрування, використовуються два ключі: публічний ключ для шифрування і приватний для дешифрування. Асиметричні алгоритми, як-от RSA (Rivest-Shamir-Adleman), DSA (Digital Signature Algorithm) та алгоритми на основі еліптичних кривих, є більш безпечними з точки зору передачі даних через незахищені канали.

Водночас вони є значно повільнішими у порівнянні із симетричними методами через складні математичні операції [7].

RSA є одним із перших, широко використовуваних асиметричних шифрів для безпечної передачі даних. Користувач RSA створює та публікує відкритий ключ на основі двох великих простих чисел разом із допоміжним значенням. Прості числа тримаються в секреті. Повідомлення може зашифрувати будь-хто за допомогою відкритого ключа, але розшифрувати їх може лише той, хто знає закритий ключ. Через те, що всі асиметричні шифри значно повільніші за якісні симетричні, то вони використовуються для шифрування ключів симетричного алгоритму. Саме тому самий популярний варіант для обміну повідомленнями є зв'язкою RSA + AES: формується ключ AES, цей ключ шифрується за допомогою відкритого ключа RSA, відправляється кореспонденту, той його розшифровує за допомогою свого закритого ключа, після цього буде можливість передавати зашифровані ключем AES повідомлення.

Основні відмінності симетричного алгоритму AES, та симетричного алгоритму RSA надав в таблиці 1.1:

Таблиця 1.1

Порівняння двох найбільш популярних алгоритмів шифрування

Характеристика	Симетричний алгоритм AES	Асиметричне алгоритм RSA
1	2	3
Ключі	Один і той самий ключ для шифрування та дешифрування	Два ключі: публічний для шифрування і приватний для дешифрування
Швидкість	Висока швидкість, особливо з великими обсягами даних	Значно повільніше через складні математичні операції
Проблеми	Необхідність безпечного обміну ключами	Складність, вимагає більше обчислювальних потужностей
Довжина ключа	128, 192, 256 біт	Зазвичай 2048-4096 біт

Закінчення таблиці 1.1

1	2	3
Використання	Більше підходить для шифрування великих обсягів даних	Більше підходить для шифрування симетричних ключів для їх безпечної передачі
Стійкість до атак	Стійкий до атак на перебір, при належній довжині ключів, квантово-стійкий	Стійкий до атак на перебір, при належній довжині ключів, не є квантово-стійким
Приклади використання	Інструменти архівування та стиснення, шифрування файлів, шифрування файлових систем, шифрування дисків/розділів, шифрування зберігання, безпека зв'язку в локальних мережах тощо	Цифрові підписи та сертифікати, захищені протоколи зв'язку, безпечний обмін ключами тощо

Квантова криптографія: загрози та нові можливості

Досягнення в галузі квантових обчислень суттєво вплинули на розвиток криптографії. Квантові комп'ютери здатні виконувати певні обчислення, як-от факторизація великих чисел або дискретне логарифмування, значно швидше, ніж класичні комп'ютери. Це ставить під загрозу традиційні криптографічні алгоритми. Саме тому, поки ще квантові комп’ютери не настільки потужні, щоб злами з їх використанням були чимось розповсюдженим, криптографи розробляють нові алгоритми, щоб підготуватися до Y2Q (стратегія, суть якої полягає в довгостроковому зберіганні зашифрованих даних, які зараз не має можливості розшифрувати, але при майбутніх можливих проривах у технологіях дешифрування, чи підвищенні потужності апаратного забезпечення може з’явитися можливість їх розшифрувати) [8].

Однак, більшість сучасних симетричних криптографічних алгоритмів вважаються відносно безпечними проти атак з використанням квантових комп'ютерів. Хоча квантовий алгоритм Гровера (квантовий алгоритм для перебору) дійсно прискорює атаки на симетричні шифри, подвоєння розміру ключа може ефективно блокувати ці атаки. Так вважається, що AES-256 є квантово-стійким, він має квантову стійкість, на рівні стійкості AES-128 проти традиційних (неквантових) атак. AES-192 не вважається квантово-стійкими через менший розмір ключа [9].

Transport Layer Security

Важливе значення має шифрування даних під час їхнього передавання через мережі, бо саме тому дані вразливі до атаки Man-in-the-Middle, коли зловмисник перехоплює трафік між двома сторонами і може змінювати чи читати передані дані. Одним із найбільш поширених протоколів для цього є TLS (Transport Layer Security), що забезпечує шифрування трафіку між клієнтом та сервером. TLS широко використовується для електронної пошти, обміну миттєвими повідомленнями, в протоколі HTTPS для захисту вебсайтів. SSL/TLS забезпечує має такі особливості:

- Автентифікація, конфіденційність, цілісність. Основною особливістю SSL/TLS - захист даних при передачі за допомогою шифрування. Є можливість використовувати автентифікацію сервера та клієнта для підтвердження ідентичності сторін. Також даний протокол забезпечує цілісність даних за допомогою імітовставки. Тому, окрім захисту від розголошення даних за допомогою шифрування, протокол безпеки TLS можна використовувати для захисту від маскарадних атак, атак типу "людина посередині".
- Сумісність. SSL/TLS працює з більшістю веб-браузерів, включаючи Microsoft Internet Explorer і Netscape Navigator, а також з більшістю операційних систем і веб-серверів, включаючи операційну систему

Microsoft Windows, UNIX, Novell, Apache (починаючи з версії 1.3), Netscape Enterprise Server і Sun Solaris.

Із недоліків можна відмітити, що використання даного протоколу, як і будь якого іншого шифрування, помітно збільшують навантаження на центральний процесор. Навантаження залежить від частоти встановлення з'єднань та їх тривалості. Найбільше навантаження виникає саме під час встановлення з'єднань [10]. Щоб зменшити навантаження використовується відновлення рукостискання, найбільш вимогливими є операції з відкритими ключами, тому TLS забезпечує механізм відновлення сеансів. Відновлені сеанси реалізуються за допомогою ідентифікаторів сеансів, або мандатів сеансів. Окрім підвищення продуктивності, відновлені сеанси також можна використовувати для єдиного входу [11].

IPsec

IPsec – це набір протоколів для захисту мережевих з'єднань, який аутентифікує та шифрує пакети даних, що проходять через IP мережі. IPv4 мав замало забезпечень безпеки, тому був розроблений IPsec, який є моделлю того самого мережевого рівня (третій рівень мережевої моделі OSI). IPsec має наступні функції та протоколи:

- Authentication Header (AH) забезпечує цілісність даних та автентифікацію джерела даних, а також забезпечує захист від атак на модифікацію IP-заголовків і атак повторення. AH містить значення перевірки цілісності в частині даних автентифікації заголовка, яке зазвичай побудовано на основі стандартних криптографічних хеш-алгоритмів, наприклад SHA-1.
- Encapsulating Security Payload (ESP) забезпечує конфіденційність, цілісність даних, автентифікацію джерела даних, службу захисту від повторення і обмежену конфіденційність трафіку.
- Internet Security Association and Key Management Protocol (ISAKMP) забезпечує платформу для автентифікації та обміну ключами, надає можливості: ручний ввід ключів, що були передані допомогою певного

захищеного каналу; Internet Key Exchange; Kerberized Internet Negotiation of Keys; DNS IPSECKEY. Використовується ISAKMP, щоб створити спільні атрибути безпеки між двома об’єктами мережі для підтримки безпечного зв’язку (Security Association, або SA) з пакетом алгоритмів і параметрів, необхідних для роботи AH та/або ESP.

Зазвичай AH та ESP використовуються окремо, хоча іноді можливо використовувати їх разом. Протоколи IPsec AH і ESP можуть бути реалізовані в транспортному режимі хост-хост, а також у режимі мережевого тунелювання:

Транспортний. У транспортному режимі AH IP-пакет лише незначно змінюється, щоб включити новий заголовок AH між IP-заголовком і корисним навантаженням, також відбувається перемішування коду протоколу, який пов’язує різні заголовки разом. Це перемішування протоколу потрібне, щоб вихідний IP-пакет міг бути відновлений на іншому кінці: заголовки IPsec після отримання та перевірки видаляються, а вихідний тип протоколу (TCP, UDP тощо) зберігається у заголовку IP. Коли пакет приходить до місця призначення та проходить перевірку автентифікації, заголовок AH видаляється, а поле Proto=AH у заголовку IP замінюється збереженим next=протоколом. Це повертає пакет до початкового стану, і його можна буде доставити до процесу очікування.

При використанні ESP у транспортному режимі відбувається інкапсуляція лише корисного навантаження пакету. Оригінальний IP-заголовок залишається на місці (за винятком перетасованого поля «Протокол»), тобто адреси джерела та призначення залишаються незмінними.

Режим тунелю. Подібно до транспортного режиму, пакет «запечатується» за допомогою значення перевірки цілісності для запобігання модифікації під час пересилання та автентифікації відправника. Різниця полягає в інкапсуляції повного IP-заголовок та корисне навантаження, що дозволяє адресам джерела та призначення відрізнятися від адрес охоплюючого пакета: це дозволяє сформувати тунель. Коли пакет у тунельному режимі надходить до місця

призначення, він проходить ту саму перевірку автентифікації, що й будь-який пакет типу AH, при проходженні перевірки заголовки IP та AH видаляються, що фактично відтворює оригінальний пакет.

ESP в режимі тунелю інкапсулює весь пакет в зашифровану оболонку. Також це відрізняється від AH тим, що спостерігач не має можливості визначити, у тунельному, чи у транспортному цей трафік, а все через те, що факт, що цей режим є тунельним – частина зашифрованого корисного навантаження, яке не являється читабельним для спостерігача [12].

OpenPGP – один із самих поширених стандартів шифрування електронної пошти. OpenPGP походить від PGP – сімейства програмних систем, що були розроблені Філом Циммерманном. OpenPGP забезпечує конфіденційність і цілісність даних для повідомлень і файлів даних за допомогою відкритого ключа та/або симетричного шифрування та цифрових підписів. Він надає формати для кодування та передачі зашифрованих та/або підписаних повідомлень. Крім того, OpenPGP надає функціональні можливості для кодування та передачі ключів і сертифікатів.

Основні можливості:

OpenPGP може поєднувати шифрування з симетричним ключем та шифруванням з відкритим ключем для забезпечення конфіденційності. При використанні відкритих ключів спочатку об'єкт шифрується за допомогою алгоритму шифрування з симетричним ключем. Кожен симетричний ключ використовується лише один раз для одного об’єкта. Новий сеансовий ключ генерується як випадкове число для кожного об’єкта, який і є сеансом. Оскільки він використовується лише один раз, сеансовий ключ прив’язується до повідомлення та передається разом з ним. Щоб захистити ключ, він шифрується відкритим ключем одержувача.

Автентифікація за допомогою цифрового підпису. На стороні відправника формується підпис з хеш-суми повідомлення та додається до повідомлення.

Потім, на стороні отримувача формується хеш отриманого повідомлення та перевіряється за допомогою прикладеного підпису.

Стиснення. OpenPGP підтримує стиснення, але не всі розробники додають можливість стиснення, але зазвичай хоча б відновлення даних присутнє.

Перетворення на Base64. Базовим представленням OpenPGP для зашифрованих повідомлень, підписів, ключів та сертифікатів – потік довільних октетів. Для транспортування необроблених двійкових октетів OpenPGP через канали, які не є безпечними для передачі необроблених двійкових даних, використовується кодування цих двійкових октетів. Необроблений 8-розрядний двійковий потік октетів можна перетворити на потік друкованих символів ASCII за допомогою кодування base64 у форматі ASCII Armor. Коли OpenPGP кодує дані в ASCII Armor, він розміщує спеціальні заголовки навколо даних закодованих base64, щоб OpenPGP міг відновити дані пізніше [13].

Також, хоча OpenPGP й створено для використання як шифрування, так і підписів, але він надає можливість використовувати лише підписування.

S/MIME

MIME (Multipurpose internet mail extension) – стандарт розширення протоколу електронної пошти, яке дозволяє обмінюватися текстом, що включає символи не тільки з системи кодів ASCII, також дозволяє додавати до листів зображення, аудіо- та відеофайли тощо.

S/MIME (Secure/Multipurpose Internet Mail Extensions) – стандарт для шифрування відкритим ключем, підписання та стиснення даних MIME. Має наступні функції:

- Підписання даних. Цифровий підпис формується таким чином: отримується хеш повідомлення, цей хеш шифрується за допомогою закритого ключа підписувача повідомлення і підпис кодуються кодуванням base64.

- Цифровий конверт. Дана функція використовуються для забезпечення конфіденційності повідомлення. Цифровий конверт складається з зашифрованого повідомлення та ключа, якщо отримувачів декілька, то копії сеансового ключа шифруються їх відкритими ключами.
- Прозоро підписані дані: формується цифровий підпис повідомлення, як і в випадку звичайного підписання, але за допомогою base64 тут кодується лише цифровий підпис. Це робиться для того, щоб отримувачі без підтримки s-MIME мали можливість переглянути хоча б вміст повідомлення, без перевірки підпису.
- Підписані дані в конверті. Дана функція є просто поєднанням перших двох функцій, при її використанні отримувач повинен буде спочатку розшифрувати повідомлення, щоб мати змогу перевірити цифровий підпис [14].

Блокчейн і криптографія

Однією з інноваційних технологій, яка тісно пов'язана з криптографією, є блокчейн. Блокчейн – розподілена база даних, що зберігає впорядкований ланцюжок блоків, який постійно довшає. Завдяки децентралізованій структурі та використанню криптографічних механізмів для захисту даних блокчейн широко застосовується для фінансових транзакцій, а також у таких сферах, як охорона здоров'я, логістика, управління активами.

Блокчейн гарантує цілісність даних через криптографічні хеш-функції, а також надає можливість перевірки кожного блоку (містить часову позначку, хеш попереднього блоку та дані транзакцій, подані як хеш-дерево). Для забезпечення цілісності хешу всього блоку включається у наступний блок в ланцюжку. Однак, ця технологія не ідеальна, вона має проблеми з масштабованістю та енергетичною ефективністю, тому тривають активні дослідження щодо її оптимізації.

Гомоморфне шифрування

Ще однією цікавою технологією в галузі шифрування є гомоморфне шифрування. Цей метод дозволяє виконувати обчислення над зашифрованими даними без їхнього розшифрування. Така можливість відкриває можливості для зберігання та обробки конфіденційної інформації в хмарних середовищах, де необхідно гарантувати приватність даних навіть під час їхньої обробки. Наприклад, компанії, що надають хмарні послуги, можуть обробляти зашифровані дані користувачів без необхідності їхнього розшифрування, зберігаючи конфіденційність інформації. Щоб вважатися алгоритмом з гомоморфними властивостями він повинен підтримувати всього дві операції: додавання та множення. Unpadded RSA є одним з перших алгоритмів з гомоморфними властивостями [15].

Проте гомоморфне шифрування є дуже ресурсозатратним і поки що не застосовується широко не тільки через високу обчислювальну складність, але й обмежений функціонал та відсутності стандартів.

1.3. Огляд програмних рішень для шифрування ресурсів

З розвитком технологій і збільшенням обсягу оброблюваних даних, зростає і необхідність захисту інформації від несанкціонованого доступу. Програми для шифрування ресурсів мають основну роль у захисті конфіденційної інформації як для індивідуальних користувачів, так і для організацій. Тому я роздивлюся декілька програмних рішень для шифрування даних.

1) ESET Endpoint Encryption

ESET Endpoint Encryption – це платне пропрієтарне програмне забезпечення для шифрування, створене британською компанією DESlock, яку словацька компанія ESET придбала в 2015 році [16].

- Відповідність стандартам

Для захисту даних підприємств різних розмірів рішення використовує підтверджене стандартами FIPS 140-2 (стандарт комп'ютерної безпеки уряду США, який використовується для затвердження криптографічних модулів. Початкова публікація відбулася 25 травня 2001 року та була оновлена 3 грудня 2002 року) шифрування 256-біт AES (є варіантом блокового шифру Rijndael, Rijndael – це сімейство шифрів з різними розмірами ключів і блоків. Для AES NIST вибрав три члени сімейства Rijndael, кожен з розміром блоку 128 біт, але трьома різними довжинами ключа: 128, 192 і 256 біт.).

- Кросплатформна підтримка

Дозволяє управляти шифруванням на робочих станціях Windows та вбудованим шифруванням macOS (FileVault) за допомогою однієї інформаційної панелі.

- Захист даних під час передачі

Шифрування електронних листів і вкладень та обмеження доступу до змінних носіїв для певних користувачів дозволяє захистити дані під час передачі та запобігти їх витоку за межі компанії.

- Підтримка різних типів шифрування

Рішення підтримує функції повнодискового шифрування, шифрування файлів та папок (не можна шифрувати папки на мережевому диску або копіювати зашифровані папки на мережевий диск, однак є можливість створити віртуальний зашифрований диск на мережевому диску), змінних носіїв та електронних листів.

- Шифрування тексту та буфера обміну

Шифрування текстового вікна повністю або частково у веб-браузері, базі даних або веб-пошті.

- Відсутність необхідності в додатковому обладнанні

TPM для використання повнодискового шифрування не вимагається.

- Централізоване управління

Повний контроль за ліцензуванням, технічними характеристиками програмного забезпечення, політикою конфіденційності та ключами шифрування.

- Шифрування листів та вкладень

Легкість надсилання та отримання зашифрованих електронних листів та вкладень, що доступна через надбудову для Outlook.

- Віртуальні диски та зашифровані архіви

Можливість створити безпечний зашифрований том на ПК чи іншому пристрої, або зашифровану копію цілого дерева каталогів та файлів. Віртуальні диски – це зашифровані контейнери, які надають доступ до файлів на віртуальних дисках через зіставлену літеру диска.

2) VeraCrypt

VeraCrypt – це безкоштовна утиліта з відкритим вихідним кодом для шифрування на льоту. Програмне забезпечення може створити віртуальний зашифрований диск, який працює як звичайний диск, але всередині файлу. Він також може зашифрувати розділ або (у Windows) весь пристрій зберігання даних за допомогою автентифікації перед завантаженням [17].

VeraCrypt використовує AES, Serpent, Twofish, Camellia та Kuznyechik як шифри. Для додаткової безпеки доступні десять різних комбінацій каскадних алгоритмів:

1. AES-Twofish
2. AES-Twofish-Serpent
3. Camellia-Kuznyechik
4. Camellia-Serpent
5. Kuznyechik-AES
6. Kuznyechik-Serpent-Camellia
7. Kuznyechik-Twofish
8. Serpent-AES

9. Serpent-Twofish-AES
10. Twofish-Serpent

Криптографічні хеш-функції, доступні для використання у VeraCrypt, це BLAKE2s-256, SHA-256, SHA-512, Streebog і Whirlpool.

Режим блокового шифрування VeraCrypt – XTS. Він генерує ключ заголовка та вторинний ключ заголовка (режим XTS) за допомогою PBKDF2 із 512-бітовою солі. За замовчуванням вони проходять від 200000 до 500000 ітерацій, залежно від використовуваної базової хеш-функції та від того, чи є це системним, чи несистемним шифруванням. Користувач може налаштувати його, щоб починати від 2048 і 16000 відповідно.

У версії 1.24 VeraCrypt була додана опція шифрування ключів і паролів у оперативній пам’яті у версіях Windows x64, а також опцію видалення всіх ключів шифрування з пам’яті під час підключення нового пристрою. Короткий опис:

Під час запуску Windows драйвер VeraCrypt виділяє область пам’яті розміром 1 МБ. Якщо це не вдається, ми збільшуємо розмір пристрою на два, доки розподіл не вдасться (мінімальний розмір становить 8 КБ). Усі ці змінні розміщуються в не вивантаженому просторі пам’яті ядра.

Потім ця область пам’яті заповнюється випадковими байтами, згенерованими CSPRNG (генератор псевдовипадкових чисел, який не генерує такі послідовності, які не здатен відрізнити від повністю випадкових послідовностей жоден ефективний алгоритм за поліноміальний час. Іншими словами, жоден статистичний тест не буде здатен відрізнити отриману послідовність псевдовипадкових чисел від насправді випадкової послідовності) на основі ChaCha20.

Генеруються два випадкових 64-розрядних числа, HashSeedMask і CipherIVMask.

Для кожного головного ключа тому алгоритм шифрування оперативної пам’яті отримує унікальний ключ із комбінації області пам’яті та унікальних

значень, витягнутих із пам’яті для шифрування. Це забезпечує окремий ключ для кожної зашифрованої області пам’яті. Використання залежних від розташування ключів і IV запобігає легкому вилученню головних ключів із дампів пам’яті.

Головні ключі розшифровуються для кожного запиту, що вимагає швидкого алгоритму розшифрування. Для цього використовується ChaCha12.

Після монтування тому його головні ключі негайно шифруються за описаним алгоритмом.

Для кожного запиту введення-виведення для тому головні ключі розшифровуються лише протягом цього запиту, а потім безпечно стираються.

Після відключення тому зашифровані головні ключі надійно видаляються з пам’яті.

Під час завершення роботи або перезавантаження Windows область пам’яті, виділена під час запуску, безпечно стирається.

VeraCrypt не використовує переваги Trusted Platform Module (TPM). VeraCrypt FAQ дослівно повторює негативну думку оригінальних розробників TrueCrypt. Розробники TrueCrypt вважали, що виключною метою TPM є «захист від атак, які вимагають від зловмисника прав адміністратора або фізичного доступу до комп’ютера». Але зловмисник, який має фізичний або адміністративний доступ до комп’ютера, може обійти TPM, наприклад, встановивши апаратний реєстратор натискань клавіш, захопити вміст пам’яті та отримати ключі, видані TPM.

Як і його попередник TrueCrypt, VeraCrypt підтримує створення одного «прихованого тому» в іншому томі. Версії VeraCrypt для Windows можуть створювати та запускати приховану зашифровану операційну систему. Документація VeraCrypt перераховує способи, якими функції заборони прихованого тому можуть бути скомпрометовані (наприклад, стороннім програмним забезпеченням, яке може витікати інформацію через тимчасові файли або через мініатюри), а також можливі способи уникнути цього.

3) BitLocker

BitLocker – це програма для шифрування дисків, розроблена Microsoft і інтегрована в операційну систему Windows. Основні особливості: шифрування всього диска, підтримка TPM (Trusted Platform Module) для зберігання ключів шифрування, простота використання для кінцевих користувачів [18].

За замовчуванням він використовує алгоритм Advanced Encryption Standard (AES) у ланцюжку блоків шифру (CBC) або режим «xor-encrypt-xor (XEX) на основі налаштованої кодової книги з крадіжкою зашифрованого тексту» (XTS) із 128-бітним або 256-бітним ключем. CBC не використовується на всьому диску, він застосовується до окремих секторів.

Перед запуском Windows використовуються функції безпеки, реалізовані у складі обладнання та вбудованого ПЗ пристрою, включаючи TPM та безпечне завантаження:

TPM – це мікросхема, призначена для надання основних функцій, пов'язаних із безпекою, насамперед із використанням ключів шифрування. BitLocker прив'язує ключі шифрування до TPM, щоб гарантувати, що пристрій не було змінено, доки система перебуває в автономному режимі.

UEFI – це програмоване завантажувальне середовище, яке ініціалізує пристрої та запускає завантажувач операційної системи. Специфікація UEFI визначає процес автентифікації для виконання вбудованого ПЗ, цей процес називається безпечним завантаженням.

Безпечне завантаження блокує ненадійні вбудовані програми та завантажувачі (підписані або непідписані) від запуску в системі. За замовчуванням BitLocker забезпечує захист цілісності для безпечного завантаження за допомогою вимірювання PCR довіреного платформного модуля. Несанкціоноване вбудоване програмне забезпечення EFI, завантажувальна програма EFI або завантажувач не можуть запуститися й отримати ключ BitLocker

Для захисту від зловмисних атак скидання (Коли платформа перезавантажується або вимикається, вміст енергозалежної пам’яті (RAM) не втрачається відразу. Без електричного заряду для збереження даних у пам’яті дані почнуть розпадатися. Протягом цього періоду є короткий проміжок часу, протягом якого зловмисник може вимкнути або перезавантажити пристрій, а потім швидко ввімкнути його, щоб завантажити програму, яка створює копію вмісту пам’яті. Ключі шифрування та інші секрети можуть бути легко скомпрометовані за допомогою цього методу) BitLocker використовує пом’якшення атак скидання TCG, також відомий як біт MOR (запит на перезапис пам’яті), перед тим, як видобувати ключі в пам’ять.

4) FileVault 2

FileVault 2 – це програма для шифрування дисків, розроблена Apple і інтегрована в операційну систему Mac OS. Для блокового шифрування використовується алгоритм XTS-AES-128, оптимізований для 512-байтних блоків [19].

Основною ідеєю FileVault 2 є простота використання для звичайних користувачів. Має вбудовану мережу безпеки відновлення, зазвичай ключі відновлення зберігаються на серверах Apple. FileVault 2 відображає ключ відновлення та пропонує користувачеві зберегти ключ серверах Apple при вмиканні. Ключ відновлення є 120-бітним та може складатися з усіх літер англійського алфавіту та цифр від 1 до 9, генерується він за допомогою генератора псевдовипадкових чисел.

Адміністрування. При використанні FileVault 2 в організації у адміністраторів є механізм для надання авторизованого доступу до томів кінцевих користувачів у випадках, коли потрібно отримати доступ без знання паролю, наприклад, коли користувач забув його. Для відновлення використовується FVMI (FileVault Master Identity).

Для будь-якого зашифрованого тому є рядок ключів шифрування, що використовуються для захисту, доступу, скидання паролів і відновлення.

Керування ключами передбачає керування та використання як ключів шифрування, так і ключів відновлення. Є три рівні ключів шифрування та два варіанти ключів відновлення:

- Ключ шифрування томів (VEK). 256-бітний VEK використовується менеджером томів на найнижчому рівні для роботи з 512-байтними логічними блоками. Кожен VEK генерується випадковим чином для кожного незалежного логічного тому. Оскільки доступ до даних на тому залежить від його VEK, він має залишатися постійним протягом усього часу існування тому.
- Ключ шифрування ключів (KEK). Під час ініціалізації зашифрованого тому генерується випадковий симетричний проміжний ключ шифрування, який шифрує VEK і зберігається в метаданих менеджера томів. Цей ключ дозволяє змінювати похідні ключі шифрування (DEK) незалежно як один від одного, так і від VEK, та змінювати VEK незалежно від DEK. Після того, як VEK зашифровано за допомогою KEK, він зберігається в метаданих цього тому.
- Похідний ключ шифрування (DEK). Щоб розшифрувати та доступу до зашифрованого тому потрібно розшифрувати два попередні ключа, саме для цього використовується похідний ключ шифрування. Будь-який DEK можна змінити не впливаючи ні на KEK, ні на VEK.

Порівняння вище згаданих рішень по ключовим характеристикам надав в таблиці 1.2:

Таблиця 1.2

Порівняння існуючих рішень

Характеристика	ESET Endpoint Encryption	VeraCrypt	BitLocker	FileVault 2
1	2	3	4	5
TPM	Підтримує	Не підтримує	Підтримує	Не підтримує

Продовження таблиці 1.2

1	2	3	4	5
Тип програми	Платна, пропрієтарна	Безкоштовна, з відкритим кодом	Платна (інтегроване в деякі версії Windows), пропрієтарна	Платне (інтегрована в macOS), пропрієтарна
Шифрування	AES-256, 3DES, або Blowfish	AES-Twofish, AES-Twofish-Serpent, Camellia-Kuznyechik, Camellia-Serpent, Kuznyechik-AES, Kuznyechik-Serpent-Camellia, Kuznyechik-Twofish, Serpent-AES Serpent-Twofish-AES Twofish-Serpent	AES-128 або AES-256 (CBC або XTS)	AES-XTS-128
Режими шифрування	Повне шифрування диска, файлів, папок і змінних носіїв. Підтримка шифрування електронної пошти, тексту і буфера обміну	Шифрування розділу або всього диска, можливість прихованого тому	Повне шифрування диска	Повне шифрування диска
Шифрування в пам’яті	-	Так, за допомогою ChaCha12	Ні	Ні

Закінчення таблиці 1.2

1	2	3	4	5
Управління ключами	Централізоване та локальне управління ключами	Локальне управління ключами	Локальне управління ключами	Централізоване та локальне управління ключами з можливістю зберігання ключів відновлення на серверах Apple
Простота використання	Гнучке налаштування для підприємств, централізоване управління	Потребує більш глибокого налаштування, більше можливостей, для досвідчених користувачів	Інтегроване у Windows, мінімальна кількість налаштувань	Інтегроване у macOS, мінімальна кількість налаштувань
Кроспла-тформність	Windows 10, 8, 8.1, 7, Vista, XP, Server 2003 - Server 2016, macOS Mojave та новіші, iOS	Windows 10 та новіші, Windows Server 2016-Windows Server 2019, macOS 12 та новіші, Linux, FreeBSD	Enterprise та Ultimate версії Windows Vista та Windows 7; Pro та Enterprise версії Windows 8 та 8.1; Windows Embedded Standard 7 та Windows Thin PC; Windows Server 2008 та новіші; Pro, Enterprise, та Education версії Windows 10; Pro, Enterprise, та Education версії Windows 11	OS X 10.7 Lion та новіші

1.4. Висновки до першого розділу

В даному розділі було аналізовано сучасний стан забезпечення безпеки в інформаційних мережах.

Були визначені основні загрози, як теперішні, так і до яких ще не так і близько.

Розглянув аспекти, які характеризують стан інформаційної безпеки в сучасних інформаційних мережах.

Були описані тенденції розвитку технологій для забезпечення інформаційної безпеки, а саме технологій шифрування.

Також було згадано про роль банківської сфери в цьому, нормативних актів, а в особливості регламенти Європейського Союзу.

Під кінець були досліджені існуючі на ринку продукти, що дозволяють використовувати шифрування для захисту інформації.

РОЗДІЛ 2
ОПИС ОБРАНИХ МЕТОДІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ

2.1 Загальна структура застосунку

З огляду на те, що застосунок буде включати в себе не одну функцію, а буде комплексним, то є умовно поділити його на декілька частин, модулів. Програмний застосунок буде складатися з кількох основних модулів, які виконують певні функції для забезпечення безпеки, зручності та ефективності роботи з конфіденційними даними, кожен модуль відповідає за окремий етап обробки даних або взаємодію з користувачем. Основні компоненти застосунку повинні включати:

2.1.1 Модуль шифрування

Даний модуль є основним в застосунку. Він використовує сучасні алгоритми для забезпечення надійної захищеності файлів та інших ресурсів. Основні функції цього модуля включають:

- Шифрування даних та розшифрування: Модуль використовує ключі з постійної пам’яті та симетричний алгоритм шифрування для перетворення даних у зашифрований вигляд чи навпаки, для їх розшифрування.
- Хешування: Використання функцій хешування для безпечного зберігання паролів та генерації ключів з них.

2.1.2 Модуль керування ключами

Відповідає за створення, зберігання та керування криптографічними ключами, що використовуються для шифрування та розшифрування даних, та зберігаються в постійній пам’яті. Всі ключі зберігаються у зашифрованому вигляді для забезпечення їх безпеки. Основні функції цього модуля включають:

- Генерація ключів: Створення випадкових або похідних ключів для шифрування. Ключі можуть бути генеровані на основі даних користувача за допомогою відповідної функції першого модуля, або з використанням криптографічно стійких випадкових значень.
- Зберігання ключів: Ключі шифрування зберігаються в зашифрованому вигляді, щоб треті обличчя не змогли їх використати, навіть у випадку викрадення.

2.1.3 Біометричний модуль

Забезпечує інтеграцію технологій біометричних методів для забезпечення додаткового рівня безпеки автентифікації та шифрування даних. Основні функції цього модуля включають:

- Збір біометричних даних: Модуль дозволяє користувачам додавати до системи свої біометричні дані.
- Аналіз біометричних даних: Дозволяє обробляти зібрані біометричні дані, для подальшого безпечного зберігання та використання.
- Аутентифікація з використанням біометричних даних: Під час входу в систему, користувач має використовувати не лише пароль та логін, але й біометрію.
- Зберігання біометричних даних: Модуль має механізми захисту біометричних даних, щоб уникнути їх несанкціонованого доступу або зловживання. Дані зберігаються в обробленому та зашифрованому вигляді.

2.2 Шифрування

2.2.1 Вибір алгоритму шифрування

Оскільки програма використовується локально, то застосування повільних асиметричних алгоритмів шифрування не є оптимальним варіантом – потрібно використовувати симетричні алгоритми шифрування.

Розгляну та порівняю деякі найбільш відомі симетричні алгоритми шифрування (сортування від найшвидшого до найповільнішого) [20]:

Таблиця 2.1

Порівняння найпопулярніших симетричних алгоритмів шифрування

Назва	Ключова довжина, біт	Розмір блоку, біт	Раундів	Структура
AES	128, 192, 256	128	10, 12, 14	мережа замін-перестановок
BlowFish	32-448	64	16	Мережа Фейстеля
IDEA	128	64	8.5	Схема Лая–Массі
3DES	112, 168	64	48	Мережа Фейстеля

З таблиці 2.1 можна зрозуміти, що AES є найкращим варіантом, оскільки він не тільки найшвидший з-поміж перелічених, але й має найбільший запас безпеки, та й взагалі являється стандартом.

2.2.2 Advanced Encryption Standard

Advanced Encryption Standard (AES), або ж Rijndael – симетричний алгоритм блочного шифрування, має розмір блоку 128 біт, ключі можуть бути від 128 до 256 біт.

Ввід та вивід відбувається за допомогою 128-бітних блоків. Ввід ключів відбувається за допомогою послідовностей 128, 192, чи 256 бітів. Основною одиницею в алгоритмі є 8 біт (байт). Байтові значення для зручності часто представляють в шістнадцятковій системі числення, де кожен символ – пів байта.

Робота з блоками відбувається за допомогою State. State – це проміжний результат блокового шифру AES, який є двовимірним масивом байтів із чотирма рядками та Nb (Кількість стовпців, в складі State, де кожен стовпець – 32-розрядне слово (послідовність чотирьох байтів), стандарт N_b= 4) стовпцями:

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

Таблиця 2.2

Залежність параметрів шифру AES від розміру ключа

	Довжина ключа	N_k	N_r	Розмір блоку	N_b
AES-128	128	4	10	128	4
AES-192	192	6	12	128	4
AES-256	256	8	14	128	4

де N_k – кількість 32-розрядних слів, що містять ключ;

N_r – кількість раундів;

N_b – кількість стовпців, в складі State.

Порядок роботи AES:

Повідомлення розбивається на блоки по 128 біт, які перетворюються на State.

Keyexpansion. З основного ключа генеруються $4*(N_r+1)$ 32-розрядних слів.

AddRoundKey. Додавання ключа початкового раунду.

Потім, від першого, до передостаннього раунду виконуються такі операції над State:

Subbytes. Оборотне, нелінійне перетворення стану, у якому таблиця підстановки (S-блок), застосовується до кожного байта в State;

Shiftrows. Перетворення State, при якому байти в останніх трьох рядках State циклічно зсуваються на кількість позицій, що залежить від індексу рядка;

Mixcolumns. Перетворення State, кожна колонка приймається за многочлен четвертого степеня $b(x)=b_3x^3+b_2x^2+b_1x+b^0$, де кожен байт є елементом поля Галуа $GF(2^8)$. Кожна колонка множиться по модулю x^4+1 на фіксований многочлен $a(x)=3x^3+x^2+x+x+2$;

Addroundkey. Поєднання ключа з State.

Після виконання цих чотирьох операцій над необхідними раундами три з них виконуються над останнім раундом, а саме: Subbytes, Shiftrows, Addroundkey, тобто не виконується лише Mixcolumns [21].

Для прискорення роботи шифру також можна об’єднати Subbytes, Shiftrows та Mixcolumns, за допомогою попередньо обчислених таблиць, тобто замість виконання цих операцій заміни, зсуву та множення, виконується пошук значень у таблицях. Але в такому випадку потрібно більше пам’яті, саме для цих таблиць [22].

2.2.3 Безпека AES

Хоча станом на 2024 рік навіть для великих організацій з неймовірними обчислювальними потужностями атаки перебором займають занадто багато

часу та коштують ледве не всіх грошей світу, навіть в випадку використання 128-бітного ключа.

Але окрім перебору існують і інші типи атак:

Атака з пов'язаними ключами. Ця атака спрямована на AES-256, використовує всього два пов’язані ключі та 2^{39} операцій для відновлення повного 256-бітного ключа 9-раундової версії, 2^{45} операцій для 10-раундової версії, 2^{70} разів для 11-раундової версії. Тобто ця атака вимагає, щоб криптоаналітик мав доступ до відкритих текстів, зашифрованих кількома ключами, пов’язаними певним чином, працює лише проти 11 раундів, коли повний AES-256 має 14. З чого можна зробити висновок, що вона не представляє суттєвої загрози [23].

Атака відновлення ключа на повний AES. Для її реалізації зловмисник має пару, або декілька пар відкритого та зашифрованого тексту. Атака є швидша за атаку перебором приблизно в чотири рази. Але все ще недостатньо швидка, для самого вдалого варіанту атаки відновлення ключа на AES-256 потрібно $2^{254.3}$ операцій [24].

Атаки з використанням побічних каналів. Даний тип атак дозволяє відновити ключі шифрування AES, використовуючи вразливості не самого алгоритму, а як він був реалізований. В 2016 році була публічно представлена атака цього типу на AES-128, яка може відновити 128-бітний ключ за 6–7 блоків відкритого, чи зашифрованого тексту. Хоча може здаватися, що використання AES-192, чи AES-256 повністю зводить це нанівець, але це не зовсім так. Зі збільшенням довжини ключа до 256 біт ускладнює атаку приблизно у 6 разів, що є незначною різницею з огляду на все зростаючу доступність обчислювальних ресурсів [25].

Квантові атаки. В даному типі для атак використовуються квантові комп’ютери. Як уже було сказано в першому розділі, наразі до квантових атак стійким вважається лише AES-256. Хоча сфера квантових обчислень розвивається доволі швидко, надійність AES-256 вона поки не ставить під

загрозу. Так, згідно з дослідженням 2019 року, для квантової атаки перебором на AES потрібні тисячі логічних кубітів з виявленням і виправленням помилок, якщо бути точним, то для AES-256 потрібно 6681 логічних кубітів, які складаються з 3347181-42096981 фізичних кубітів. Самий потужний квантовий комп'ютер 2023 року складається з 1225 фізичних кубітів [26, 27].

Таким чином, можна зробити висновок, що наразі AES-256 є більш ніж безпечним шифром, бо відомі атаки, або не мають практичної цінності, так як проводяться в вельми специфічними умовами, які зазвичай неможливо створити в реальних умовах, або потребують занадто багато ресурсів. В моєму програмному рішенні цей алгоритм буде використовуватися для шифрування як ресурсів, так і ключів, так і біометричних даних.

2.3 Хешування

2.3.1 Вибір алгоритму хешування

Хешування – це процес перетворення даних будь-якого розміру на значення фіксованої довжини за допомогою спеціальних математичних алгоритмів, відомих як хеш-функції. Результат цього перетворення називається хешем, або хеш-значеням.

Криптографічна хеш-функція – хеш-функція, яка повинна відповідати ряду вимог, які є критичними для криптографії, а саме:

- стійкість до зіткнень, тобто, що знаходження двох таких повідомлень, що перетворюються хеш-функцією в однаковий хеш не представляється можливим;
- неможливість перетворення хешу в оригінальне повідомлення;
- відсутність можливості зміни повідомлення без зміни хешу;
- відносна легкість обчислення;

Використовується хешування для самих різних завдань: перевірка цілісності даних, зберігання паролів, створення цифрових підписів тощо. В

моєму випадку воно використовується для безпечного зберігання паролів користувачів, для генерування ключів шифрування.

В таблиці 2.3 розгляну та порівняю деякі найбільш популярні криптографічні алгоритми хешування (сортування від найшвидшого до найповільнішого) [28]:

Таблиця 2.3

Порівняння найпопулярніших симетричних алгоритмів хешування

Назва	Довжина хеша, біт	Розмір блоку	Раундів	Надійність
SHA-2	224, 256, 384, 512	512, 1024	64, 80	Надійний
SHA-1	160	512	80	Знайдено зіткнення [29]
MD5	128	512	4	Знайдено зіткнення [30]
Tiger	160, 128, 192	512	24	Надійний

Отже, можна дійти висновку, що SHA-2 та Tiger найкращі варіанти, але якщо враховувати, що Tiger не тільки повільніший на сучасних системах за SHA-2, більш старий та при цьому менш вивчений, то найкращим варіантом буде використання SHA-2.

2.3.2 SHA-2

SHA-2 – сімейство криптографічних хеш-функцій, що був розроблений Агентством національної безпеки в 2001, як заміна SHA-1, який ставав все більш потенційно ненадійним. Включає в себе чотири повноцінних функцій (SHA-224, SHA-256, SHA-384, SHA-512) та дві урізані версії SHA-512 (SHA-512/224, SHA-512/256).

В своєму програмному застосунку буду використовувати SHA-512 (для безпечного зберігання паролів користувачів) та SHA-256 (для генерації 256-бітного ключа шифрування AES-256). Основні етапи їх роботи:

2.3.2.1 Попередня обробка

Доповнення повідомлення:

До до кінця вхідного повідомлення довжиною l біт додається один одинарний біт та така кількість k нульових бітів, де k – найменший невід'ємний розв'язок рівняння $l+1+k \equiv 448 \, mod \, 512$ (для SHA-256), чи $l+1+k \equiv 896 \, mod \, 1024$ (для SHA-512).

Додавання довжини повідомлення:

Після доповнення додається 64-бітний блок (у випадку SHA-512 він буде 128-бітним), який дорівнює l.

2.3.2.2 Парсинг повідомлення

Під парсингом мається на увазі розбиття повідомлення на N m-бітних блоків.

Для SHA-256 повідомлення та його доповнення розбиваються на N 512-бітних блока, $M^{(1)}, M^{(2)}, M^{(3)}, \ldots, M^{(N)}$. А самі блоки розбиваються на шістнадцять 32-бітних слова $M_0^{(i)}, M_1^{(i)}, M_2^{(i)}, \ldots, M_{15}^{(i)}$.

Для SHA-512 повідомлення та його доповнення розбиваються на N 1024-бітних блока, $M^{(1)}, M^{(2)}, M^{(3)}, \ldots, M^{(N)}$. А самі блоки розбиваються на шістнадцять 64-бітних слова $M_0^{(i)}, M_1^{(i)}, M_2^{(i)}, \ldots, M_{15}^{(i)}$.

2.3.2.3 Початкове хеш-значення

Перед обчисленням хешу необхідно встановити початкове значення хешу. Розмір і кількість слів у якому залежить від розміру дайджесту повідомлення.

Для SHA-256 початкове хеш-значення $H^{(0)}$, має складатися з наступних восьми 32-бітних слів, які є першими 32 бітами дробових частин квадратного кореня перших 8 простих чисел (2, 3, 5, 7, 11, 13, 17, 19):

$H_0^{(0)}$=6a09e667

$H_1^{(0)}$=bb67ae85

$H_2^{(0)}$=3c6ef372

$H_3^{(0)}$=a54ff53a

$H_4^{(0)}$=510e527f

$H_5^{(0)}$=9b05688c

$H_6^{(0)}$=1f83d9ab

$H_7^{(0)}$=5be0cd19

Для SHA-512 початкове хеш-значення $H^{(0)}$, має складатися з наступних восьми 64-розрядних слів, які є першими 64 бітами дробових частин квадратного кореня перших 8 простих чисел (2, 3, 5, 7, 11, 13, 17, 19):

$H_0^{(0)}$=6a09e667f3bcc908

$H_1^{(0)}$=bb67ae8584caa73b

$H_2^{(0)}$=3c6ef372fe94f82b

$H_3^{(0)}$=a54ff53a5f1d36f1

$H_4^{(0)}$=510e527fade682d1

$H_5^{(0)}$=9b05688c2b3e6c1f

$H_6^{(0)}$=1f83d9abfb41bd6b

$H_7^{(0)}$=5be0cd19137e2179

2.3.2.4 Обчислення хешу

SHA-256 використовує шість логічних функцій. Кожна з цих функцій працює з 32-бітними словами (x, y, z):

$$Ch(x,y,z)=(x \wedge y) \oplus (\neg x \wedge z) \tag{2.1}$$

$$\mathrm{Maj}(x,y,z)=(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{2.2}$$

$$\sum_{0}^{256}(x)=ROTR^{2}(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \tag{2.3}$$

$$\sum_{1}^{256}(x)=ROTR^{6}(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \tag{2.4}$$

$$\sigma_{0}^{[256]}(x)=ROTR^{7}(x) \oplus ROTR^{18}(x) \oplus ROTR^{3}(x) \tag{2.5}$$

$$\sigma_{1}^{[256]}(x)=ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus ROTR^{10}(x) \tag{2.6}$$

У SHA-512 перші дві функції 2.1 та 2.2 такі самі, а чотири інших 2.3-2.6 трохи відрізняються, бо працюють вони з 64-бітними словами:

$$\sum_{0}^{512}(x)=ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \tag{2.7}$$

$$\sum_{1}^{512}(x)=ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x) \tag{2.8}$$

$$\sigma_{0}^{[512]}(x)=ROTR^{1}(x) \oplus ROTR^{8}(x) \oplus ROTR^{7}(x) \tag{2.9}$$

$$\sigma_{1}^{[512]}(x)=ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus ROTR^{6}(x), \tag{2.10}$$

де $ROTR^{n}(x)$ – це операція повороту праворуч, в якій x – w-бітне слово, а n – таке ціле число, що $0 < n < w$.

Окрім функцій також необхідно сказати про використовувані константи. SHA-256 використовує 64 констант K, які є першими 32 бітами дробових частин кубічного кореня перших 64 простих чисел. В свою чергу SHA-512 використовує 80 констант K, які є першими 64 бітами дробових частин кубічного кореня перших 80 простих чисел.

Використовуються вище зазначені функції 2.1-2.10 для перетворення кожного блоку $M^{(1)}, M^{(2)}, M^{(3)}, \ldots, M^{(N)}$:

1) Підготовка розкладу. Суть підготовки полягає в розширенні 16-слівного, 512-бітного вхідного блоку до масиву W_t із 64 слів (додавання обчислюється за модулем 2^{32}) :

$$W_t = \begin{cases} M_t^{(i)} & 16 \le t \le 15 \\ \sigma_1^{[256]}(W_{t-2}) + W_{t-7} + \sigma_0^{[256]}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases};$$

у випадку SHA-512 блок 1024-бітний, тому має 80 слів в W_t та додавання обчислюється за модулем 2^{64}:

$$W_t = \begin{cases} M_t^{(i)} & 16 \le t \le 15 \\ \sigma_1^{[512]}(W_{t-2}) + W_{t-7} + \sigma_0^{[512]}(W_{t-15}) + W_{t-16} & 16 \le t \le 79 \end{cases}$$

2) Ініціалізація робочих змінних a, b, c, d, e, f, g і h, присвоєння їм значень поточних хешів:

a=$H_0^{(i-1)}$

b=$H_1^{(i-1)}$

c=$H_2^{(i-1)}$

d=$H_3^{(i-1)}$

e=$H_4^{(i-1)}$

f=$H_5^{(i-1)}$

g=$H_6^{(i-1)}$

h=$H_7^{(i-1)}$

3) Цикл стискання (зображений на рис. 2.1) за схемою Меркле-Демґарда, що відбувається для кожного значення t від 0 до 63 (або до 79 для SHA-512):

$$T_1 = h + \sum_1^{256}(e) + Ch(e,f,g) + K_t^{[256]} + W_t$$

$$T_2 = \sum_0^{256}(a) + Maj(a,b,c)$$

Для SHA-512: $T_1 = h + \sum_1^{512}(e) + Ch(e,f,g) + K_t^{[512]} + W_t$

Для SHA-512: $T_2 = \sum_0^{512}(a) + Maj(a,b,c)$

$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

4) Після циклу стиснення іде обчислення і-тих проміжних хешів:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

Після перетворення всіх блоків $M^{(1)}, M^{(2)}, M^{(3)}, \ldots, M^{(N)}$ за допомогою конкатенації отримуємо 256-бітний хеш (у випадку SHA-512 – 512-бітний): $H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$ [31].

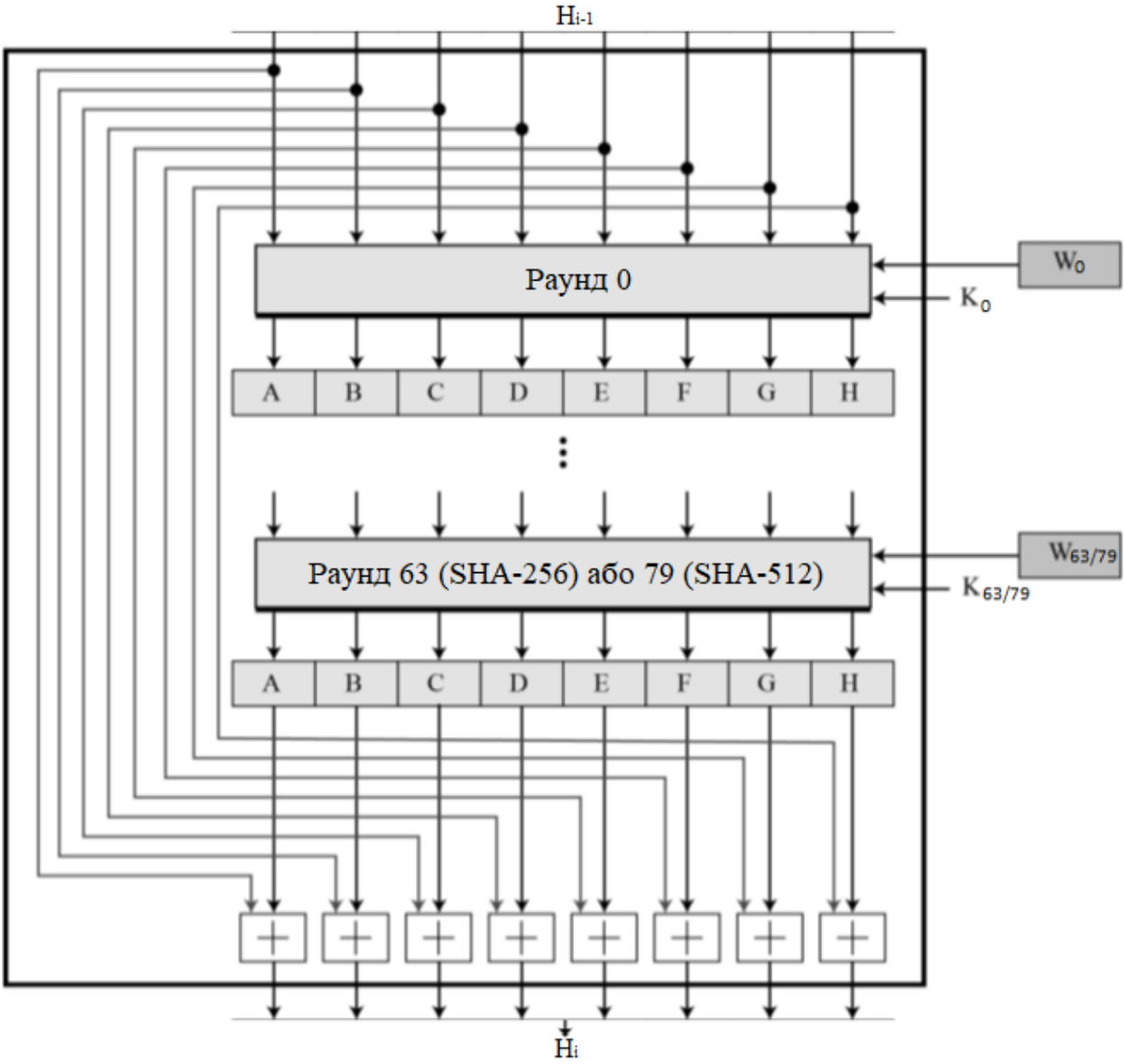

Рис. 2.1 Алгоритм стиснення

2.3.2.5 Безпека SHA-2

За останні роки було чимало атак на SHA-2, але жодна з атак не може бути застосована на повну хеш-функцію. Останній відомий здобуток криптоаналізу сімейства SHA-2 був опублікований в 2023 році, там розглядалися атаки зіткнення.

SHA-256: Раундів в якому на три більше, ніж в попередньому кращому результаті, а саме 31 з 64 раундів. При цьому має складність $2^{49.8}$.

SHA-512: В якому раундів на чотири більше, ніж в попередньому кращому результаті, а саме 31 з 80 раундів. При цьому має складність $2^{115.6}$ [32].

Отже, бачимо, що існуючі атаки на обмежені версії SHA-2, наразі не є серйозною загрозою для практичного застосування цього сімейства хеш-функцій. Причинами цьому є висока складність обчислень та урізана кількість раундів, що робить ці теоретичні атаки далекими від практичного застосування.

2.4 Генерація ключів

Найбільш розповсюджений та використовуваний спосіб для генерації ключів – це генерація за допомогою генераторів псевдовипадкових чисел (ГПВЧ). Генератор псевдовипадкових чисел – криптографічний алгоритм для отримання високоякісних випадкових бітів із джерела ентропії. Джерело невизначеності або джерело ентропії, забезпечується генераторами випадкових чисел (ГВЧ).

Генератори псевдовипадкових чисел (ГПВЧ) вирішують проблему, з якою стикаються під час генерування випадковості, надійно виробляючи багато штучних випадкових бітів із кількох справжніх випадкових бітів. Наприклад, ГВЧ, який перетворює рухи миші у випадкові біти, перестане працювати, якщо ви припините рухати мишею, тоді як ГПВЧ завжди повертає псевдовипадкові біти, коли їх запитують. ГПВЧ покладаються на ГВЧ, але поводяться інакше: ГВЧ створюють справжні випадкові біти відносно повільно з використанням аналогових джерел, без гарантії високої ентропії. Коли ГПВЧ швидко створює біти випадкового вигляду за допомогою цифрових джерел, з максимальною ентропією. Тобто, ГПВЧ перетворюють певну кількість ненадійних випадкових бітів у довгий потік надійних псевдовипадкових бітів, які можна використовувати в криптографії.

Існують як криптографічні, так і звичайні ГПВЧ. У криптографії необхідно використовувати криптографічно стійкі генератори псевдовипадкових чисел (КСГПВЧ), які забезпечують стійкість до

прогнозування. Це означає, що навіть якщо зловмисник отримає доступ до частини згенерованої послідовності, йому не вдасться передбачити інші числа в послідовності або вихідні параметри (зерно) [33]. Саме тому в програмному застосунку буде використовуватися КСГПВЧ. Не так давно CryptGenRandom була стандартною функцією КСГПВЧ в операційній системі Windows, була частиною Microsoft CryptoAPI. Але тепер для цього використовується функція BCryptGenRandom, що включена в Cryptography API: Next Generation (CNG), що замінив застарілий CryptoAPI [34].

2.5 Робота з біометрією облич

Робота будь-якої системи розпізнавання обличчя складається з декількох етапів:

- локалізування обличчя на зображенні;
- вирівнювання обличчя, що враховує позу, освітлення тощо;
- зчитування;
- порівняння зі значенням в базі даних.

2.5.1 Методи розпізнавання

Геометричний підхід. Такі системи виявляють ознаки обличчя, аналізують відносне їх розташування, розмір і форму очей, носа, вилиць та підборіддя тощо. Після аналізу вони порівнюються з аналогічними значеннями іншого зображення обличчя. Мають високу швидкість роботи та можуть працювати навіть з низькою якістю зображень, але вони менш ефективні при зміні освітлення та куту повороту голови.

Фотометричний підхід. Фотометричні методи працюють із текстурою обличчя, аналізуючи тональні і колірні особливості. Використовуючи математичні моделі, як-от метод головних компонент (МГК) або лінійний дискримінантний аналіз (ЛДА) вони створюють образ обличчя. Ці методи

працюють краще при зміні виразів обличчя, але можуть бути менш точними при поганому освітленні.

Підхід з використанням графів. Полягає в створенні сітки, де вершини графа відповідають певним точкам на обличчі (наприклад, навколо очей, носа, губ і т.д.), а ребра графа описують зв'язки між цими точками, що можуть представляти, наприклад, просторові відстані між різними частинами обличчя. На етапі розпізнавання два графи порівнюються: алгоритм намагається деформувати тестовий граф так, щоб його вершини і ребра стали максимально схожими на еталонний граф. Тестовий граф деформується, тобто його вершини рухаються в різні напрямки, щоб зменшити різницю між ознаками в його точках і точках еталонного графа. Це робиться для кожної вершини, поки різниця не стане мінімальною.

Методи на основі глибокого навчання. Системи, що мають за основу дані методи використовують згорткові нейронні мережі для створення складних моделей, здатних розпізнавати обличчя з високою точністю. Під час створення моделей їх навчають на певному наборі готових прикладів, в яких мережа має знайти ключові ознаки та побудувати взаємозв’язки між ними. Використовуючи ці взаємозв’язки вона розпізнає нові, невідомі об’єкти.

Трьохвимірне розпізнавання. Техніка тривимірного розпізнавання обличчя використовує Використовують тривимірні моделі обличчя, створені на основі глибини, форми та розміщення окремих частин обличчя. Перевагою тривимірного розпізнавання є те, що воно не залежить від змін освітлення, на відміну від інших методів. Воно також дозволяє розпізнавати обличчя під різними кутами, зокрема в профіль. Потребують пристроїв для побудови тривимірних зображень, що помітно дорожче звичайної камери.

Тепловізори. Іншим методом збору даних для розпізнавання облич є використання теплових камер, які можуть фіксувати форму голови, ігноруючи аксесуари, такі як окуляри, шапки чи макіяж. Теплові камери можуть знімати зображення облич навіть за умов слабкого освітлення або вночі без спалаху, що

дозволяє уникнути виявлення камери. Однак, такі системи потребуюсь тепловізорів з високою роздільною здатністю, які є значно дорожчими за звичайні камери та 3D-сканери. Точність залежить від змін температури навколишнього середовища.

Таблиця 2.4

Порівняння методів розпізнавання обличчя

Метод	Основні характеристики	Переваги	Недоліки	Типові застосування
1	2	3	4	5
Геометри-чний	Аналізує фізичні ознаки обличчя (форма, розмір, розташування очей, носа, вилиць тощо).	Висока швидкість обробки. Може працювати з низькою якістю зображень.	Чутливість до змін освітлення та кута повороту голови. Низька точність при значних змінах виразів обличчя.	Розпізнавання осіб у контрольованих умовах.
Графів	Створення сітки з точок на обличчі та зв'язків між ними.	Стійкість до змін виразів обличчя. Хороша точність навіть при низькому освітленні.	Висока складність	Розпізнавання осіб у контрольованих умовах.

Продовження таблиці 2.4

1	2	3	4	5
Трьохви-мірний	Використовує 3D-моделі обличчя, засновані на глибині та формі.	Не залежить від освітлення. Стійке до змін кута повороту голови.	Висока складність та вартість обладнання	Системи з високими вимогами до надійності. Медичні та наукові дослідження. Системи, що працюють в умовах дуже низького освітлення.
Тепловізорний	Використо-вують теплові камери для збору інформації	Працює в темряві. Ігнорує аксесуари.	Висока вартість тепловізорів з високою роздільною здатністю. Точність може залежати від змін температури навко-лишнього середовища.	Системи, що працюють в умовах дуже низького освітлення.

Закінчення таблиці 2.4

1	2	3	4	5
Фотоме-тричний	Аналізує текстуру обличчя, тональні та колірні особливості	Стійкість до змін виразів обличчя. Висока точність для зображень з хорошим освітленням.	Погана якість при низькому освітленні. Чутливість до змін кута повороту.	Розпізнавання осіб у контрольованих умовах.
Глибокого навчання	Використовує згорткові нейронні мережі для аналізу зображень обличчя.	Висока точність навіть при змінному освітленні. Може працювати з обличчями під різними кутами. Висока ефективність при аналізі виразів.	Дуже висока складність Тривалий процес навчання, для якого потребує великих наборів даних	Використання в системах відеонагляду

2.5.2 Переваги та недоліки систем розпізнавання обличчя

Переваги:

- Мінімальна взаємодія. Для сканування обличчя людині потрібно просто показати обличчя камері, не потрібно ні дотиків, ні слів, які потрібні для сканування відбитків пальців та розпізнавання голосу відповідно. Системи розпізнавання обличчя можуть здійснювати масову ідентифікацію без участі тестованої особи (що викликає стурбованість щодо порушень конфіденційності, але це вже інша тема).

- Простота впровадження. Системи розпізнавання облич можна інтегрувати в уже наявні мережі відеоспостереження без значних інвестицій в інфраструктуру. Будь-який смартфон та майже будь-який ноутбук також має камеру, якої буде буль ніж достатньо для впровадження такої системи.

Недоліки:

- Низька ефективність у деяких умовах. Якість зображень, такі як освітлення, вирази обличчя, поза тощо, можуть значно впливати на точність розпізнавання.
- Помилки: Такі системи має доволі високий рівень помилок, як першого (помилкове прийняття), так і другого (помилкове відхилення) роду, особливо з плином часу, бо обличчя людини може доволі сильно змінитися за навіть невеликий проміжок часу.

Отже, можна сказати, що є певні технічні виклики, як-то залежність від якості зображень та вплив зовнішніх факторів, але використання сучасних технологій та високоякісних камер дозволяють мінімізувати ці недоліки.

2.5.3 Біометричний модуль

Система розпізнавання обличчя повинна вміти:

a) знаходити обличчя на зображенні, що отримує з камери
b) враховувати кути
c) вміти виділяти унікальні риси обличчя (розмір роту, носу і т. ін.)
d) давати можливість зберігати інформацію про обличчя, а не його фото
e) порівнювати обличчя з даними з бази даних

2.5.3.1 Знаходження обличчя на фото

Для цього будемо використовувати метод гістограми напрямлених градієнтів, або ж HOG (histogram of oriented gradients), так як він являється одним із самих відомих та ефективних методів для цього [35].

Суть методу полягає в тому, що зовнішній вигляд і форму локального об’єкта часто можна охарактеризувати за рахунок розподілу локальних градієнтів інтенсивності або напрямках країв.

Робота HOG:

Алгоритм роздивляється кожен окремий піксель зображення, та ті, що його оточують. Робить він це для того, щоб визначити, наскільки темним є поточний піксель у порівнянні з сусідніми, а на основі цього визначається вектор, що показує, в якому напрямку зображення стає темнішим. Після проведення цієї дії для для кожного окремого пікселя на зображенні, у підсумку кожен піксель буде замінено градієнтом, тобто стрілкою що показує перехід від світлого до темного по всьому зображенню. Отримуємо ми їх для точного представлення однакових зображень, навіть якщо вони дуже сильно відрізняються яскравістю, оскільки при аналізі пікселів напряму вони будуть мати кардинально різні значення. Для кольорових зображень розраховуються окремі градієнти для кожного колірного каналу.

В результаті ми отримуємо градієнти для кожного пікселя, що занадто детально, оскільки нам потрібно бачити лише загальну схему зображення. Для воно розбивається його на квадрати 16 на 16 пікселів та підраховується, в який бік направлено більше всього окремих піксельних градієнтів – в той бік градієнт квадрату і направлений.

Щоб знайти обличчя в цьому перетвореному зображенні використовується HOG таке ж саме схематичне зображення з векторів, яке було зроблено з декількох зображень облич.

2.5.3.2 Кут зображення

Для коректного розпізнавання порівнювані обличчя повинні мати один ракурс. Щоб впевнитися в цьому потрібно мати можливості оцінювання орієнтирів обличчя та відповідне виправлення їх положення. Для цього буде використаний метод, що використовує ансамбль регресійних дерев (дерева

рішень, де цільова змінна може приймати безперервні значення, називаються регресивними), що дозволяють зменшити складність і збільшити швидкість роботи [36]. Суть метода полягає в будуванні мапи обличчя з 68-ми точок (навколо очей, губ, носа) та тренуванні алгоритму машинного навчання, щоб він міг знаходити ці точки на будь-якому обличчі.

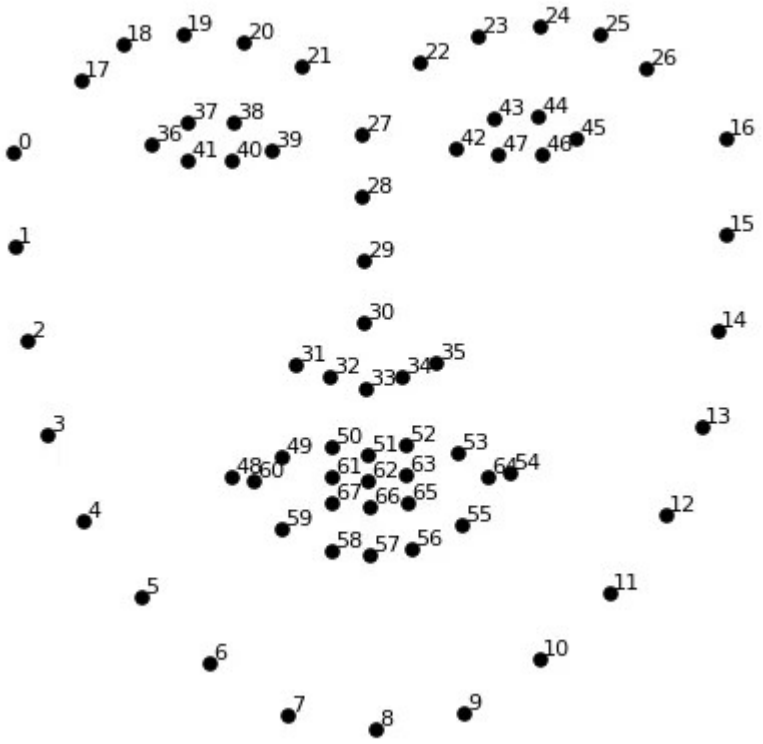

Рис. 2.2 мапа з 68-ми точок

Отримавши інформацію про розташування цих точок зображення маніпулюється таким чином, щоб очі та рот були якомога краще відцентровані. Щоб не викривити зображення (і в результаті втратити паралельні лінії) для центрування використовуються лише базові (афінні) перетворення (переміщення, масштабування, поворот, нахил). В результаті обличчя кожного разу будуть мати приблизно однаковий ракурс.

2.5.3.3 Кодування облич

Процес порівняння облич для людей є легким та очевидним, але для комп’ютера все складніше, йому потрібно чітко сказати, що саме порівнювати. Цю проблему вирішують за допомогою різноманітних математичних моделей, але найкращий метод – дати комп’ютеру самому визначити, що потрібно порівнювати. Глибоке навчання значно краще, аніж люди, визначає, які частини обличчя важливо вимірювати. Тому варто використовувати метод на основі глибокого навчання, а якщо бути точнішим, то на згортковій нейронній мережі.

Чому саме згорткові нейронні мереж. Одними з найбільш ефективних підходів у розпізнаванні облич є використання згорткових нейронних мереж, які є удосконаленням таких моделей, як когнітрон і неокогнітрон. Цей метод досягає високих результатів у гнучкості та швидкості завдяки своїй здатності враховувати двовимірну структуру зображень (краї, текстури, форми тощо), що є важливою перевагою порівняно з багатошаровими перцептронами, які не мають такої специфічної орієнтації на просторову інформацію [37].

Основні етапи роботи згорткової нейромережі:

Поділ зображення на тайли (квадрати, плитки, підзображення): Спершу зображення розбивається на невеликі перекриваючі один одного тайли, це дозволяє обробити різні частини зображення таким чином, що нейромережа зможе ідентифікувати об'єкт незалежно від його розташування.

Конволюція: Кожен тайл проходить через однакову згорткову нейронну мережу (тобто однакові ваги використовуються для всіх тайлів), якщо щось варте уваги з’явиться на будь-якому тайлі, нейромережа позначить цей тайл. Конволюційні шари навчаються виявляти базові особливості зображення, такі як контури, кути, текстури, незалежно від їхнього положення.

Збереження результатів: Після обробки кожного тайла результати записуються у новий масив, який має таку ж структуру, як і оригінальне

зображення, але з дещо меншими розмірами. Це дозволяє зберегти інформацію про те, в яких частинах зображення були виявлені варті уваги області.

Проріджування, або децимація (Downsampling): Щоб зменшити розмір масиву та зберегти тільки найважливішу інформацію, застосовується проріджування. Найчастіше використовується метод "max pooling", коли в кожному регіоні масиву розміром 2 на 2 зберігається лише найбільше, найцікавіше значення. Це зменшує розмір, залишаючи при цьому найбільш суттєві деталі.

Фінальне передбачення: Зменшений масив передається у повнозв'язну нейронну мережу, яка використовує цю інформацію для остаточного рішення, наприклад, чи є на зображенні те, на зображенні чого ми її тренували.

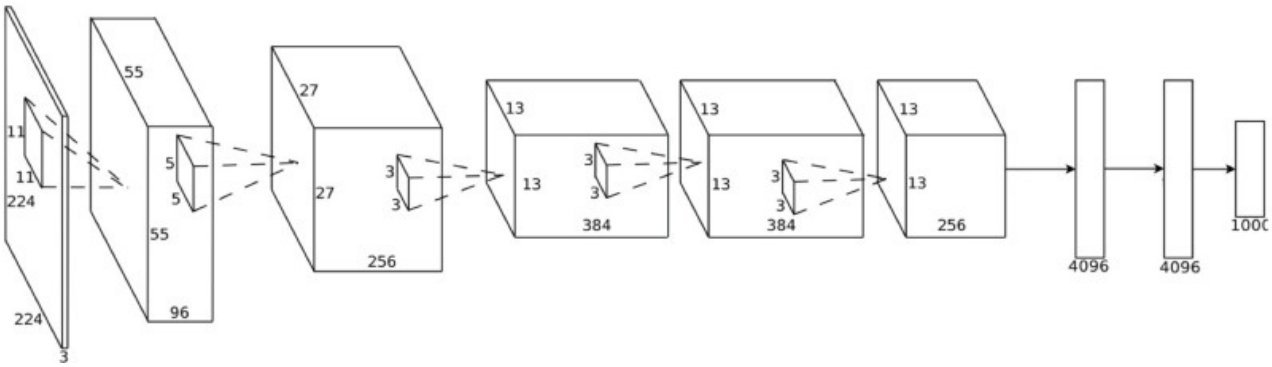

Рис. 2.3 Схема згорткової нейронної мережі

На рис. 2.3 відбувається наступне: вхідне зображення має розмірність 224 на 224 пікселя та три колірних каналів. Двічі застосовується згортка та максимальне проріджування, застосовується згортка ще 3 рази, застосовують максимальне проріджування, а потім два повнозв'язні шари. Останній шар – класифікатор, що виводить розподіл ймовірностей між 1000 мітками класів.

У нашому випадку з обличчям тренування складається з: завантаження навчального зображення обличчя, завантаження іншого зображення тієї самої людини, завантаження зображення зовсім іншої людини. Алгоритм розглядає вимірювання, які він генерує для кожного з цих трьох зображень відрегулює

нейронну мережу так, щоб вимірювання, які вона генерує для перших двох облич були трохи ближчі один до одного, а вимірювання другого та третього облич навпаки. Після повторення цього якомога більше разів з обличчями якомога більше різних людей, з десяти різних фотографій однієї людини система повинна буде видавати приблизно однакові розміри в кількості 128. Цей метод перетворення абстрактних даних (в нашому випадку – зображення обличчя) в векторні представлення – вбудовування (embedding).

2.5.3.4 Порівняння зі значеннями з бази даних

Щоб порівняти два обличчя нам просто порівняти збережені розміри з новими. Робити ми це будемо за формулою 2.11, яка простим розрахунком евклідової відстані:

$$d(x, y)=\sqrt{\sum_{i=1}^{128}(x_i - y_i)^2}, \tag{2.11}$$

де x та y – вектори двох облич.

2.6 Висновки до другого розділу

В другому підрозділі я описав модулі свого програмного застосунку та їх основні функції.

Були детально описані основні функції модулів, зокрема, шифрування, хешування та біометрії. Описав методи, що в них використовуються, чому вибрав саме їх. Так, перший підрозділ описує загальну структуру та модулі. Другий підрозділ складається з вибору алгоритму шифрування та опису його роботи. В третьому підрозділі йдеться про хешування, а саме чим воно являється, вибір алгоритму хешування та який принцип його роботи. В четвертому підрозділі йшлося про використання генератора псевдовипадкових чисел. В п’ятому ж підрозділі розповів про складові систем розпізнавання облич, методи для цього, обрав метод та описав його докладно.

РОЗДІЛ 3
РОЗРОБКА АЛГОРИТМУ ТА ПРОГРАМНОГО МОДУЛЮ ШИФРУВАННЯ

3.1 Формулювання вимог до розробки та впровадження

Вимоги до функціоналу програми:

при першому запуску програми повинен створюватися користувач адміністратора зі стандартним паролем;

користувач, дані якого є в базі користувачів повинен мати змогу увійти до системи;

якщо користувач ввів правильні дані, то повинно відкритися вікно для створення фото обличчя, щоб отримати біометричні дані, після чого або відкривається основне вікно (у випадку успіху), або відбувається повернення до вікна входу (у випадку невдачі);

кожен користувач повинен мати доступ лише до обмеженого набору функцій, відповідно до своєї ролі;

повинна бути можливість управління ключами.

3.2 Алгоритм роботи застосунку

3.2.1 Інструменти розробки

Для реалізації застосунку потрібна така мова програмування, що задовольняти наступні вимоги:

підтримує роботу з графічними інтерфейсами

має потужні бібліотеки для криптографії та шифрування

має бібліотеки для комп'ютерного зору та обробки зображень

проста робота з файловою системою

Тому обрав мову програмування Python, вона не славиться швидкістю, але ця мова програмування є гнучкою та має дуже велику базу бібліотек, особливо для роботи з криптографією та машинним навчанням.

- Області застосування Python:
- Штучний інтелект та машинне навчання;
- Веб-додатки;
- Мобільні додатки;
- Відеоігри
- Тестування на проникнення;
- Автоматизація;
- Мережеве програмування;
- Пошукова оптимізація сайтів

Основні переваги:

1) Простота

Python був розроблений, максимально нагадувати природну мову. Дуже простий синтаксис, інтерактивний інтерпретатор, велика стандартна бібліотека робить його ідеальною першою мовою програмування.

2) Портативність

Можливість, щоб Python працював на різних платформах є одним з фундаментальних елементів дизайну цієї мови. Python працює на багатьох варіантах Unix, а також на Windows, має інтерфейси для безлічі системних викликів і бібліотек, а також для різних віконних систем, розширюється на C/C++ [38].

3) Бібліотеки та спільнота

За рахунок простоти мови, її портативності, відкритості сформувалась велика спільнота розробників, які використовують дану мову програмування. В результаті чого кожного дня з’являються нові бібліотеки від спеціалістів за самих різних сфер.

4) Гнучкість

Як було сказано раніше, дана мова програмування може бути використана для самих різних задач, починаючи з простого калькулятора для невеликих чисел, закінчуючи штучним інтелектом.

З недоліків можна виділити:

1) Швидкість

А точніше її відсутність, самим головним мінусом цієї мови є те, що вона є ледве не самою повільною мовою із наразі широко використовуваних. Станом на 15.02.2023 оригінальна реалізація Python повільніша за компільовану мова програмування C в 86 разів, та в 3 рази за таку ж скриптову Lua (не LuaJIT). Але за рахунок можливості вище згаданого розширення мовами C/C++ виконання можна значно прискорювати [39].

2) Паралельне виконання

Глобальна блокування інтерпретатора, що використовується для безпечної роботи з потоками значно обмежує та ускладнює розподілення навантаження по декільком потокам, в результаті дуже часто навантаження йде на один потік процесора, скільки б їх не було в системі.

3) Інтерпретатор

Python залежить від інтерпретатора, з чого випливає, що при відсутності підтримки його середовищем можуть бути значні проблеми з розгортанням програм, які біли написані цією мовою.

В моєму випадку недоліки Python не є суттєвими, а переваги навпаки – дуже привабливі, саме тому й реалізував застосунок на цій мові програмування.

Використані бібліотеки:

- Json
- Os
- Tkinter
- OpenCV
- Face Recognition

- Numpy
- Crypto (PyCryptodome)

Json є вбудованим пакетом Python, який використовують для роботи з даними однойменного формату. В моїй програмі дані користувачів зберігаються в файлі даного формату.

Os є вбудованим пакетом Python, який використовують для роботи залежних від операційної системи функцій. Дозволяє виконувати операції з файлами та каталогами, отримувати інформацію про систему тощо.

Tkinter являється, можна сказати, стандартним пакетом Python для створення графічного інтерфейсу користувача, хоч і не офіційно.

OpenCV – бібліотека комп'ютерного зору з відкритим кодом [40].

Face Recognition – бібліотека розпізнавання облич, що дозволяє легко розпізнавати обличчя та маніпулювати рисами обличчя на зображеннях [41].

Numpy – пакет Python для наукових обчислень на Python, дозволяє працювати з великими масивами та матрицями даних.

PyCryptodome – пакет низькорівневих криптографічних примітивів на Python, дозволяє використовувати: симетричні шифри з різними режимами, асиметричні, криптографічні хеші.

3.2.2 Опис алгоритму роботи

3.2.2.1 Біометрія

В якості додаткового рівня захисту програма включає біометричний модуль, який використовує технології розпізнавання облич для автентифікації користувачів та створення ключів шифрування (у наступному підпункті 3.2.2.2). Для реалізації цього я використовуються бібліотеки:

- OpenCV, використовується для захоплення зображення з камери;
- Face recognition, використовується для аналізу та обробки зображення обличчя користувача;

- NumPy, використовується для роботи з вже обробленими біометричними даними у вигляді векторів ознак обличчя.

При першому вдалому вході до програми користувача відбувається обробка та реєстрація його біометричних даних:

1) Захоплення зображення з камери. Відбувається воно відбувається за допомогою функції def capture_face_image(). В даній функції створюється об'єкт video_capture, який і відкриває камеру для зйомки та робить фото при натисканню клавіші «q». Після створення фото воно переводиться з формату BGR (який використовує OpenCV) в RGB (з якими працює Face recognition).

2) Після захоплення зображення відбувається його обробка та аналіз. Створене зображення використовується функцією def get_face_encoding(image), де image – це і є те саме фото. Там використовуються дві функції бібліотеки Face Recognition, а саме face_locations та face_encodings. Перша функція використовує метод гістограми напрямлених градієнтів знаходить обличчя на фото та повертає масив з даними про його розташування, який використовується другою функцією, що за допомогою ансамблю регресійних дерев вирівнює позицію обличчя та використовує навчену автором згорткову нейромережу для створення вектору обличчя, що складається з 128 значень (приклад зображено на рис. 3.1).

3) Зберігання біометричних даних. За зберігання отриманого вектору відповідає функція def save_encrypted_biometric_data(user_id, password_key, face_encoding), про яку йдеться в підпункті 3.2.2.2.

4) Порівняння, яке виконується тільки у випадках повторного входу. Функція має назву def compare_faces, основою якої є linalg.norm(face_encoding_1-face_encoding_2), де norm – функція пакету linalg бібліотеки NumPy, яка обчислює евклідову відстань між збереженим та щойно зробленим векторами обличчя. При відстані менше шести десятих буде вважатися, що це вектори одного обличчя.

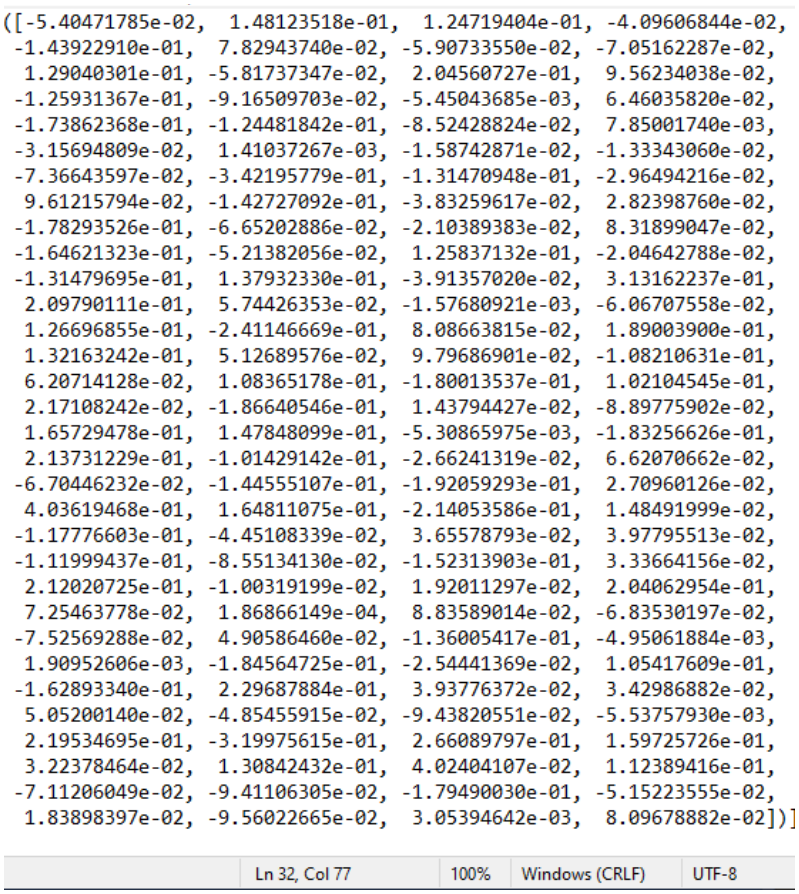

Рис. 3.1. Вектор обличчя

3.2.2.2 Криптографічні ключі

Для шифрування, про яке йдеться в наступному підпункті потрібно використовувати ключі. Програмний застосунок створює для кожного користувача по три ключі генерує три ключі за допомогою двох способів:

1) За допомогою хешування. Для генерації ключів шифрування з паролів та векторів облич використовується алгоритм хешування SHA-256, реалізоване воно за допомогою функції з пакету Hash (бібліотеки PyCryptodome) SHA256.

Використовується вона (використовуються в функціях: входу, шифрування та дешифрування головних ключів та облич, про які буде йти мова в пізніше) у вигляді рядків коду SHA256.new(password.encode()).digest() та SHA256.new(face_encoding.tobytes()).digest() розберу їх складові:

- SHA256.new() – створює хеш-об'єкт алгоритму SHA-256;

- password.encode() – перетворює в об'єкт байтів пароль користувача (password), за замовчуванням використовується кодування UTF-8;
- face_encoding.tobytes() – використовується для перетворення масиву Numpy ndarray з вектором обличчя користувача в байтову послідовність;
- digest() – повертає в вигляді байтового об'єкта дайджест вектора обличчя, цей байтовий об’єкт і є нашим 256-бітним ключем.

Також варто зауважити, доречніше про це вказати саме тут, що хешування використовується і для безпечного зберігання паролів (та, відповідно, їх перевірки), але вже для цього використовується SHA512 за допомогою однойменної функції з пакету Hash (бібліотеки PyCryptodome). Виглядає код наступним чином: SHA512.new(x.encode()).hexdigest(), де

- x – це текстовий рядок відкритого пароля користувача, це можуть бути як змінні з паролями користувачів, яких додають, так і тих, що входять до системи.
- SHA512.new() – створює хеш-об'єкт алгоритму SHA-512;
- encode() – перетворює текстовий рядок відкритого пароля користувача в об'єкт байтів, за замовчуванням також використовується кодування UTF-8;
- hexdigest() – повертає результат хешування у вигляді рядка, що містить лише шістнадцяткові цифри, який ми записуємо до бази користувачів.

2) Криптографічно стійкий генератор псевдовипадкового числа (КСГПВЧ). Використовується КСГПВЧ в функції додавання керівників департаментів. Реалізовано воно за допомогою функції з пакету Random (бібліотеки PyCryptodome) get_random_bytes(32), що повертає випадковий рядок довжиною 32 байтів, тобто 256 біт. Використовує get_random_bytes() стандартну функцію КСГПВЧ операційної системи, для віндовс це BCryptGenRandom з Cryptography API: Next Generation).

3.2.2.3 Шифрування

Шифрування використовується для безпечного збереження файлів, векторів облич та головних ключів. Шифрування відбувається алгоритмом AES-256 в режимі ланцюгування шифроблоків, цей режим розбиває даних на блоки, що залежить від попереднього блоку та вектора ініціалізації IV, що захищає як від повторень блоків, так і дозволяє легко обробляти великі обсяги даних. Використовується в наступних функціях:

1) функція зберігання зашифрованого вектора ознак (масиву Numpy) обличчя def save_encrypted_biometric_data, що на рис. 3.2 нижче. Основні елементи функції:

- першими двума рядками перевіряється існування папки для зберігання зашифрованих біометричних даних, якщо її немає, то вона створюється за допомогою makedirs з вбудованого пакету Os;
- створення об'єкта AES за допомогою однойменної функції з пакету Cipher (бібліотеки PyCryptodome) для шифрування: cipher = AES.new(password_key, AES.MODE_CBC), де
  - password_key – це ключ, який генерується із пароля користувача (створення якого описувалося в попередньому підпункті),
  - AES.MODE_CBC: вказуємо режим шифрування з ланцюгуванням шифроблоків.
- Саме шифрування: encrypted_face_encoding = cipher.encrypt(pad(face_encoding.tobytes(), AES.block_size)), де
  - tobytes() – перетворює вектор ознак обличчя в байтовий формат;
  - pad() з модуля Crypto.Util.Padding, щоб забезпечити додавання додаткових байтів для вирівнювання розміру блоку (падинг) при його некратності 16 байтам (оскільки AES-256 працює з блоками фіксованого розміру саме в 16 байт);
  - encrypt() – сама функція шифрування з модуля Crypto.Cipher.AES

- записування вектора ініціалізації та зашифрованого масиву до відповідної папки в файл, що названий іменем користувача.

```
def save_encrypted_biometric_data(user_id, password_key, face_encoding):  2 usages
    if not os.path.exists(encrypted_bio_folder):
        os.makedirs(encrypted_bio_folder)


    # Шифруємо вектор ознак обличчя
    cipher = AES.new(password_key, AES.MODE_CBC)
    encrypted_face_encoding = cipher.encrypt(pad(face_encoding.tobytes(), AES.block_size))
    # Зберігаємо зашифрований вектор ознак обличчя і IV
    with open(f"{encrypted_bio_folder}/{user_id}.npy", "wb") as f:
        f.write(cipher.iv + encrypted_face_encoding)
```

Рис. 3.2. Функція шифрування вектора

2) функція розшифрування вектора ознак обличчя def load_encrypted_biometric_data, що на рис. 3.3 нижче. Пояснення кожного рядка функції:

- відкриття файлу, в якому зберігаються зашифровані біометричні дані користувача та вектор ініціалізації;
- створення об'єкта AES для розшифрування;
- за допомогою функція з модуля Crypto.Cipher.AES decrypt() здійснюється розшифрування біометричних даних;
- оскільки під час шифрування використовувався падинг, після розшифрування ці зайві байти потрібно видаляємо за допомогою функції unpad() з модуля Crypto.Util.Padding;
- перетворюємо в масив NumPy розшифровані біометричні дані в байтовому форматі.

```python
#    Завантажує та розшифровує вектор ознак обличчя, використовуючи пароль
def load_encrypted_biometric_data(user_id, password_key):  5 usages

    # Завантажуємо зашифрований вектор і IV
    with open(f"{encrypted_bio_folder}/{user_id}.npy", "rb") as f:
        iv = f.read(16)  # IV для CBC режиму
        encrypted_face_encoding = f.read()

    # Розшифровуємо вектор ознак обличчя
    cipher = AES.new(password_key, AES.MODE_CBC, iv)
    decrypted_face_encoding = unpad(cipher.decrypt(encrypted_face_encoding), AES.block_size)

    return np.frombuffer(decrypted_face_encoding, dtype=np.float64)
```

Рис. 3.3. Функція розшифрування вектора обличчя

3) функція шифрування головних ключів доступу за допомогою ключа з обличчя def encrypt_master_key_with_face, що на рис. 3.4 нижче. Пояснення:

- генерується ключ з вектору обличчя (процес описаний в підпункті 3.2.2.2);
- все інше відбувається аналогічно функції шифрування вектора облич, тобто створюється об'єкт AES для шифрування, використовується функція encrypt(), повернення зашифрованих даних та вектора ініціалізації, щоб записати його до файлу.

```python
def encrypt_master_key_with_face(master_key, face_encoding):  2 usages
    face_key = SHA256.new(face_encoding.tobytes()).digest()
    cipher = AES.new(face_key, AES.MODE_CBC)
    encrypted_master_key = cipher.encrypt(pad(master_key, AES.block_size))
    return cipher.iv + encrypted_master_key
```

Рис. 3.4. Функція шифрування ключа обличчям

4) функція розшифрування головних ключів за допомогою ключа з обличчя def decrypt_master_key_with_face, що на рис. 3.5 нижче. Пояснення:

- генерується ключ з вектору обличчя;
- отримання вектора ініціалізації і зашифрованого головного ключа;

- аналогічно функції розшифрування вектора облич створюється об'єкт AES, використовується функції decrypt() та unpad(), в результаті чого отримуємо байтовий об’єкт ключа.

```
#    Розшифровує головний AES ключ за допомогою обличчя
def decrypt_master_key_with_face(enc_master_key, face_encoding):  4 usages
    face_key = SHA256.new(face_encoding.tobytes()).digest()
    iv = enc_master_key[:16]
    encrypted_key = enc_master_key[16:]
    cipher = AES.new(face_key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(encrypted_key), AES.block_size)
```

Рис. 3.5. Функція розшифрування ключа обличчям

5) функції шифрування (def encrypt_data) та розшифрування даних (def decrypt_data) за допомогою головного ключа , що на рис. 3.6 нижче. Пояснення:

- шифрування даних у вигляді байтового об’єкту відбувається як шифрування в попередніх функціях, тобто створюється об'єкт AES, використовується функція encrypt(), після чого йде повернення щойно зашифрованих даних та вектора ініціалізації, щоб записати його до файлу;
- шифрування даних у вигляді байтового об’єкту відбувається як розшифрування в попередніх функціях, тобто створюється об'єкт AES, використовується функція decrypt() та unpad(), в результаті використання яких отримуємо розшифровані дані у вигляді байтового об’єкту, що повертається для його подальшого збереження до файлу.

```python
#    Шифрує дані за допомогою головного ключа
def encrypt_data(data, master_key):  1 usage
    cipher = AES.new(master_key, AES.MODE_CBC)
    encrypted_data = cipher.encrypt(pad(data, AES.block_size))
    return cipher.iv + encrypted_data

#    Розшифровує дані за допомогою головного ключа
def decrypt_data(enc_data, master_key):  1 usage
    iv = enc_data[:16]
    encrypted_data = enc_data[16:]
    cipher = AES.new(master_key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(encrypted_data), AES.block_size)
```

Рис. 3.6. Функції шифрування та розшифрування даних за допомогою головного ключа

3.2.2.4 Користувачі та права доступу

1) Важливою частиною програми є робота з користувачами. Почнемо з ролей, в системі визначені наступні ролі користувачів:

- Адміністратор. Адміністратор має змогу керувати керівниками відділів, а саме: додавати, замінювати та видаляти керівників відділів. Також може змінювати самостійно свій пароль;
- Керівник. Керівник має змогу керувати звичайними користувачами, а саме: додавати, замінювати та видаляти звичайних користувачів. Також має доступ до функцій шифрування та розшифрування ресурсів. Може замінювати зашифровані ресурси;
- Звичайний користувач. Звичайний користувач має доступ тільки до функцій шифрування та розшифрування.

2) Структури даних користувачів. Користувачі зберігаються в JSON файлі (main_users.json), що містить інформацію про ім'я користувача, пароль (а точніше його хеш, що був створений за допомогою алгоритму SHA512, описання самого хешування знаходиться в підпункті 3.2.2.2), роль і департамент. Приклад бази користувачів на рис. 3.7:

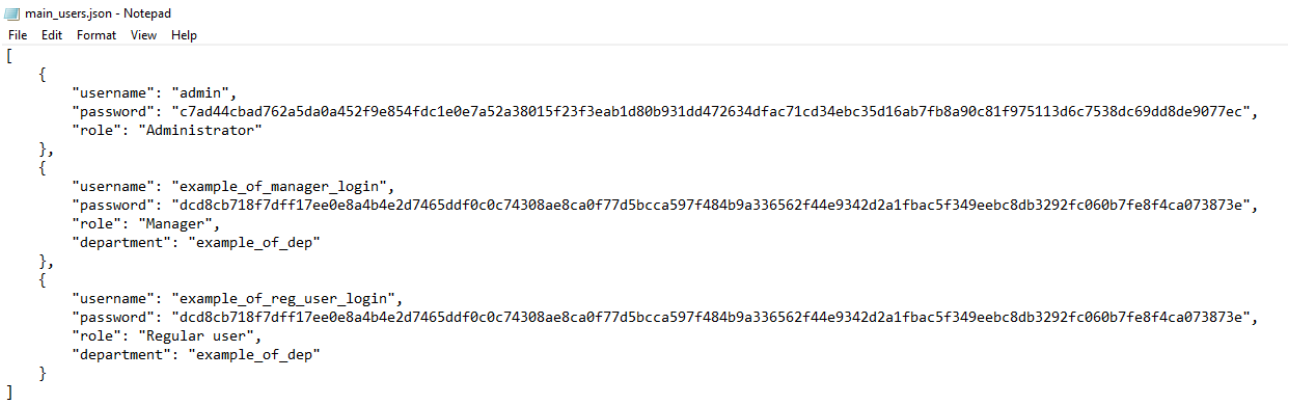

Рис. 3.7. База користувачів

3) Для читання та запису даних користувачів в базу використовуються функції (зображені на рис. 3.8) def read_user_data та def write_user_data:

- Спочатку перевіряється, чи існує файл з даними користувачів. Якщо він відсутній – автоматично створюється, в нього додається новий користувач з правами адміністратора, з логіном «admin» та паролем «admin». Якщо ж він існує, то він відкривається та дані зчитуються звідти. В будь якому разі повертається список користувачів.
- Запис до бази відбувається за допомогою другої функції. Там просто відкривається файл на читання, використовується функція dump, яка перетворює об’єкт списку з користувачами в файловий об’єкт у форматі JSON і записує його до файлу

```
def read_user_data():  6 usages
    """Зчитує дані користувачів з файлу."""
    if not os.path.exists(DATA_FILE):
        # Якщо файл не існує, створюємо його з одним стандартним користувачем
        default_users = [{'username': 'admin', 'password': SHA512.new("admin".encode()).hexdigest(), 'role': 'Administrator'}]
        write_user_data(default_users)  # Записуємо стандартного користувача в файл
        return default_users

    with open(DATA_FILE, 'r', encoding='utf-8') as file:
        try:
            return json.load(file)  # Зчитуємо дані з JSON-файлу
        except json.JSONDecodeError:
            return False

def write_user_data(users):  5 usages
    """Записує дані користувачів у файл."""
    with open(DATA_FILE, 'w', encoding='utf-8') as file:
        json.dump(users, file, ensure_ascii=False, indent=4)  # Записуємо дані
```

Рис. 3.8. Функції роботи з базою користувачів

4) Функція додавання керівників департаментів def add_manager, що зображена на рис. 3.9, працює наступним чином:

- отримується список користувачів
- перевіряється наявність в базі даних користувача з таким логіном, якщо користувач вже є, то викликається функція видалення поточного керівника вказаного департаменту def del_manager. Ця функція видаляє керівника не тільки за бази користувачів, але й зашифровані головний ключ та вектор обличчя;
- база користувачів оновлюється, щоб включити до неї нового керівника;
- якщо для вказаного департаменту вже існує головний ключ (зашифрований вектором обличчя адміністратора), то ключ: завантажується, розшифровується за допомогою біометричних даних адміністратора, щоб створити тимчасово зашифровану за допомогою пароля створюваного керівника копію. Використовуються для цього функції, що були описані в попередніх підрозділах. Не було описано лише шифрування за допомогою ключа з пароля, але воно аналогічне шифруванню за допомогою ключа з обличчя.
- якщо ключа не існує, то за допомогою функції КСГПВЧ (яка була описана в відповідному підрозділі) створюється дві копії одного головного ключа та шифруються ключем з біометричних даних адміністратора та ключем з пароля керівника;
- в будь якому випадку у нас буде два однакових ключі: один адміністратора, інший – керівника.

```
del_manager(department)
users = read_user_data()
if any(userl['username'] == username for userl in users):
    return False
#Розшифрувати скопіювати та зашифрувати паролем керівника
if os.path.exists(f"{encrypted_key_folder}/{user['username']}.key"):

    with open(f"{encrypted_key_folder}/{user['username']}.key", "rb") as f:# Читання
        encrypted_dep_key = f.read()
    dep_key = decrypt_master_key_with_face(encrypted_dep_key, saved_face_encoding)


else: #створення ключа деп., шифрув. обличчям адміна
    dep_key = get_random_bytes(32)    # Генерація випадкового головного AES ключа
    encrypted_dep_key = encrypt_master_key_with_face(dep_key, saved_face_encoding)
    with open(f"{encrypted_key_folder}/{user['username']}.key", "wb") as f:
        f.write(encrypted_dep_key)

# Створюємо ключ з пароля керівника
nmpassword_key = SHA256.new(nmpassword.encode()).digest()

# Шифрування головного ключа за допомогою пароля (керівника)
cipher = AES.new(nmpassword_key, AES.MODE_CBC)
encrypted_new_man_key = cipher.encrypt(pad(dep_key, AES.block_size))
```

Рис. 3.9. Основна частина функції додавання керівника

5) Функція додавання користувачів працює аналогічно, але де використовувалися дані та ключі адміністратора тепер використовуються такі, що належать керівнику.

6) Функція входу def login відповідає за перевірку існування користувача з вказаним іменем та порівняння відповідного йому паролю з вказаним, перевіряє або додає до системи біометричні дані користувача. Для всього цього використовує раніше описані функції, а саме: хешування паролів; обробка та реєстрації, шифрування та розшифрування біометричних даних, їх порівняння. У випадку успішного входу відкриває основне вікно.

3.3 Опис та тестування розробленого програмного застосунку

3.3.1 Робота застосунку:

При запуску програми, користувача вітає вікно входу.

У вікні входу (на рис. 3.10) присутні поля для вводу логіна та пароля, та кнопка для підтвердження вводу. Поля вводу мають надписи, що повідомляють про призначення поля, надписи зникають при натисканні на форму (та з’являються знову, якщо користувач нічого не ввів та змінив фокус на іншу форму). Пароль при введені приховується символами астериску.

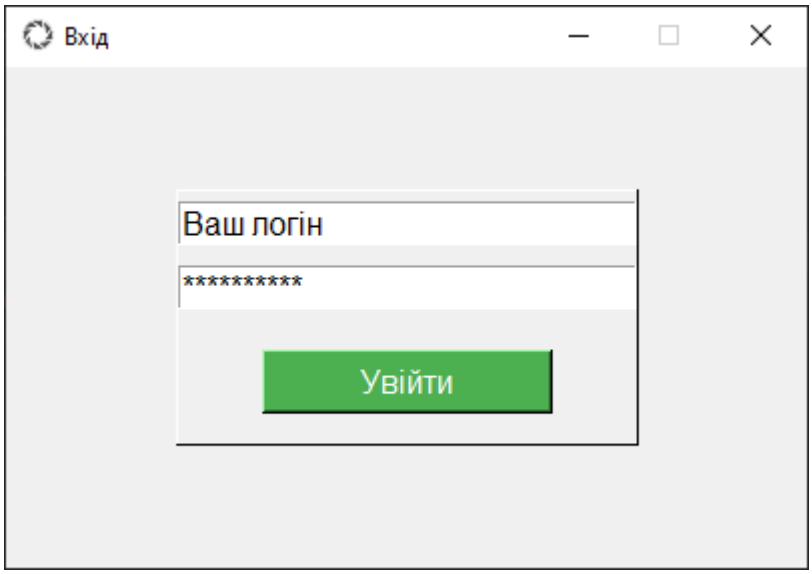

Рис. 3.10. Вікно «вхід»

Якщо база даних користувачів у файлі main_users.json відсутня, то автоматично створюється новий користувач з правами адміністратора, з логіном «admin» та паролем «admin», зображено на рис 3.11:

Рис. 3.11. Початкова база користувачів

Процес введення даних користувача зображено на рис 3.12:

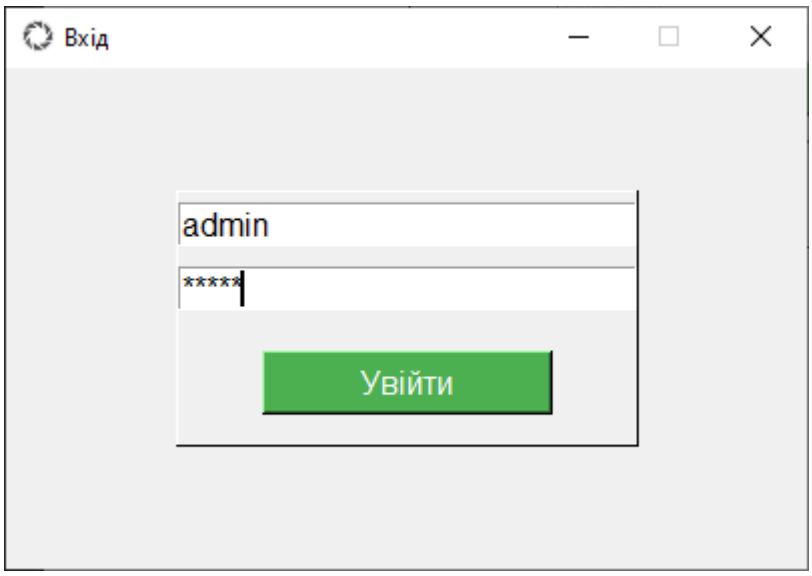

Рис. 3.12. Вікно «вхід» з введеними даними адміністратора

Після завершення вводу користувач повинен натиснути кнопку «Увійти», після чого проходить пошук користувача в базі користувачів та перевірка відповідності пароля. Якщо дані введено невірно, то вони замінюються на стандартні надписи, в разі успіху – з’явиться вікно для фотофіксації обличчя (на рис. 3.13, для демонстрації роботи з біометрією використовується віртуальна камера, що транслює фото в відеопотік. По натисканню на клавішу «q» робиться фото):

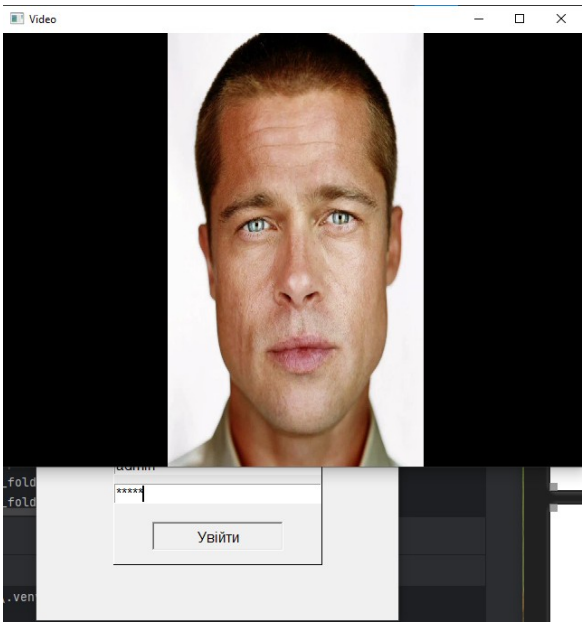

Рис. 3.13. Вікно фотофіксації

Після фотофіксації дані обличчя відповідним чином оброблюються та зберігаються за відносним шляхом: dipl\encrypted_biometrics\%логін%.npy. У цьому випадку з першим користувачем, адміністратором, файл, котрий містить зашифрований вектор обличчя називається: admin.npy (на рис. 3.14)

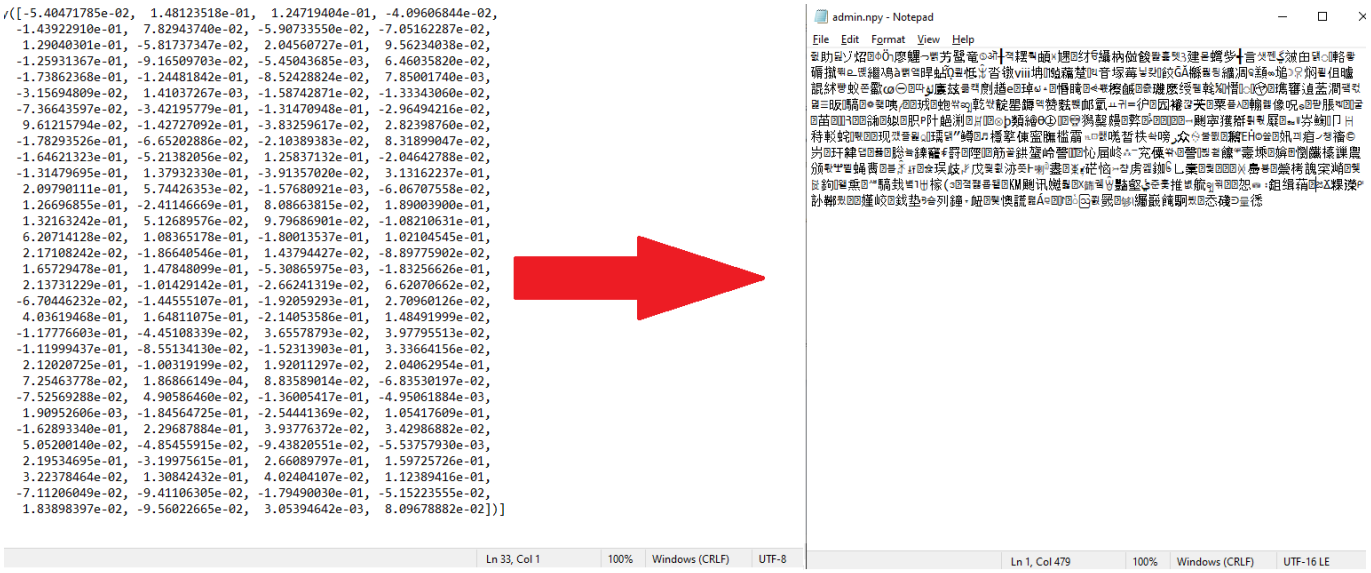

Рис. 3.14. Приклад шифрування вектору обличчя (зашифрований варіант справа)

При успішному вході користувача відкривається основне вікно програми, наповнення якого залежить від ролі користувача. Всього існує три версії основного вікна, відповідно кожній можливій ролі. Всі вони зображені на рисунках 3.15-3.17:

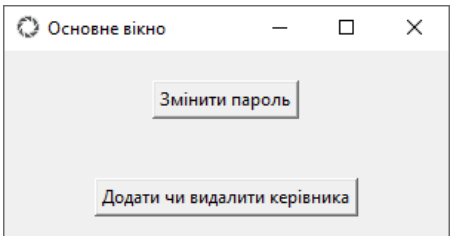

Рис. 3.15. Основне вікно адміністратора

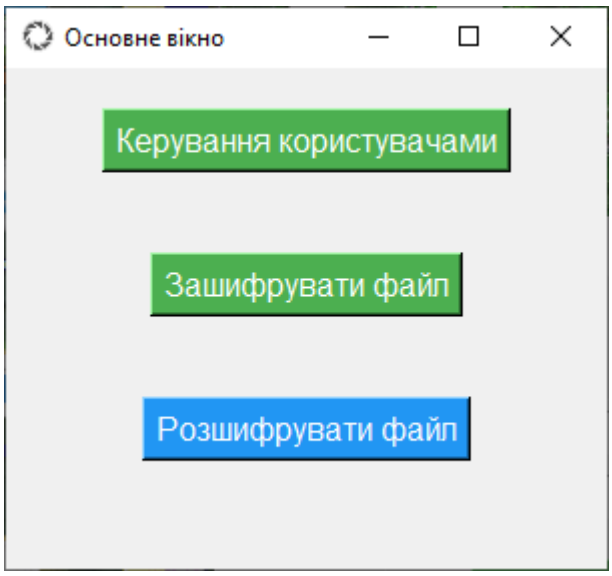

Рис. 3.16. Основне вікно керівника

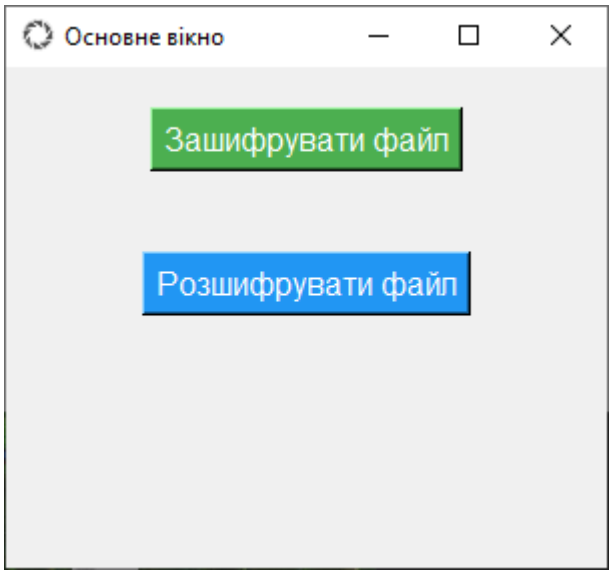

Рис. 3.17. Основне вікно звичайного користувача

Для демонстрації можливостей адміністратора додамо (на рис. 3.18) нового керівника нового департаменту, вони створяться одночасно. Підтвердження успішного створення наведені на рисунках 3.19-3.21:

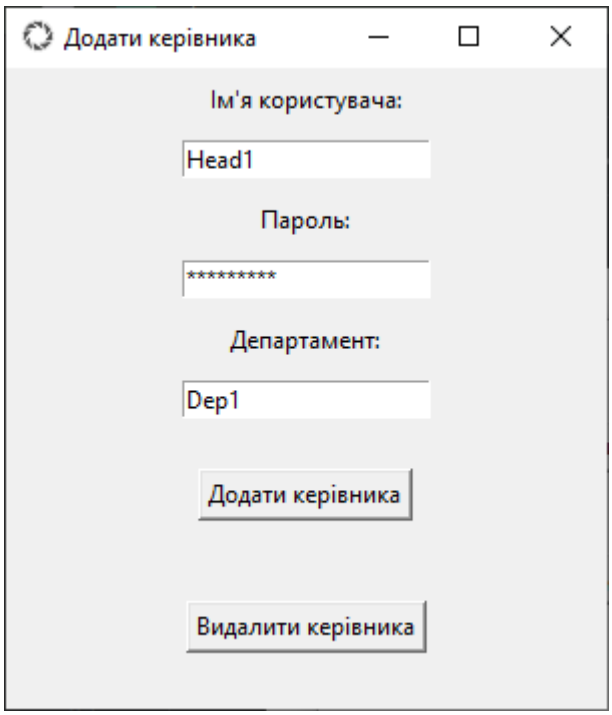

Рис. 3.18. Вікно управління керівниками

main_users.json - Notepad

File Edit Format View Help


```
[
    {
        "username": "admin",
        "password": "c7ad44cbad762a5da0a452f9e854fdc1e0e7a52a38015f23f3eab1d80b931dd472634dfac71cd34ebc35d16ab7fb8a90c81f975113d6c7538dc69dd8de9077ec",
        "role": "Administrator"
    },
    {
        "username": "Head1",
        "password": "4dff4ea340f0a823f15d3f4f01ab62eae0e5da579ccb851f8db9dfe84c58b2b37b89903a740e1ee172da793a6e79d560e5f7f9bd058a12a280433ed6fa46510a",
        "role": "Manager",
        "department": "Dep1"
    }
]
```


Рис. 3.19. Підтвердження додавання керівника нового департаменту до бази даних

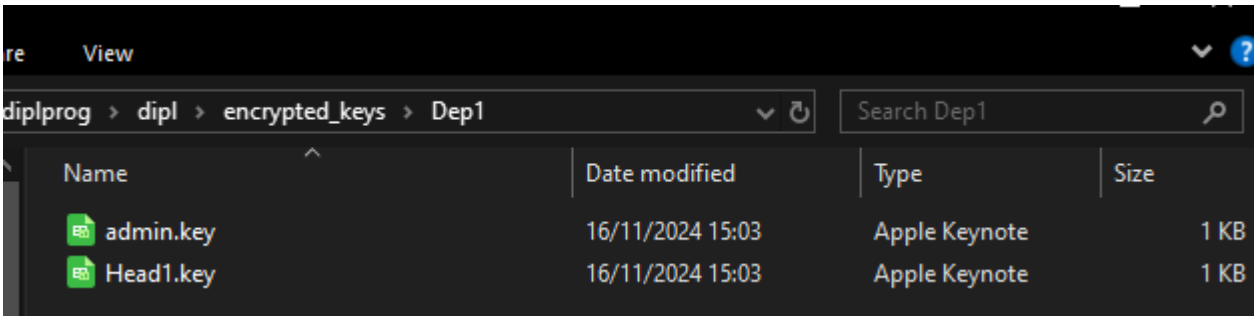

Рис. 3.20. Підтвердження створення двох копій ключа першого департаменту

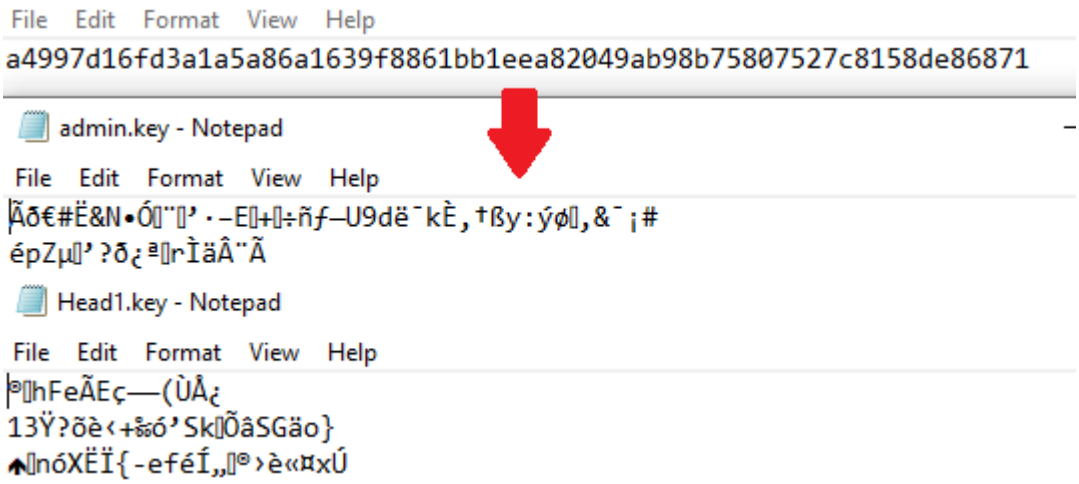

Рис. 3.21. Головний ключ до шифрування та після

На рис 3.21 зображено ключ керівника, що тимчасово, до надання біометричних даних, зашифрований парольним ключем. На рисунку 3.22-3.23 під цим, першим, керівником виконується вхід та додавання біометрії, на рис. 3.24 бачимо, що головний ключ керівника зашифрували вже біометричним ключем:

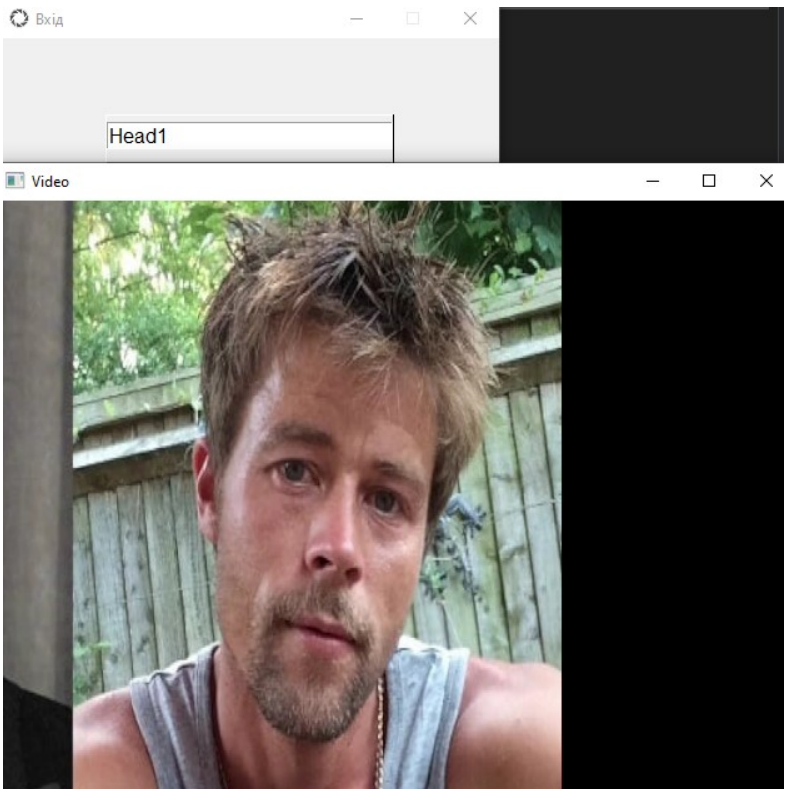

Рис. 3.22. Додавання біометрії керівника

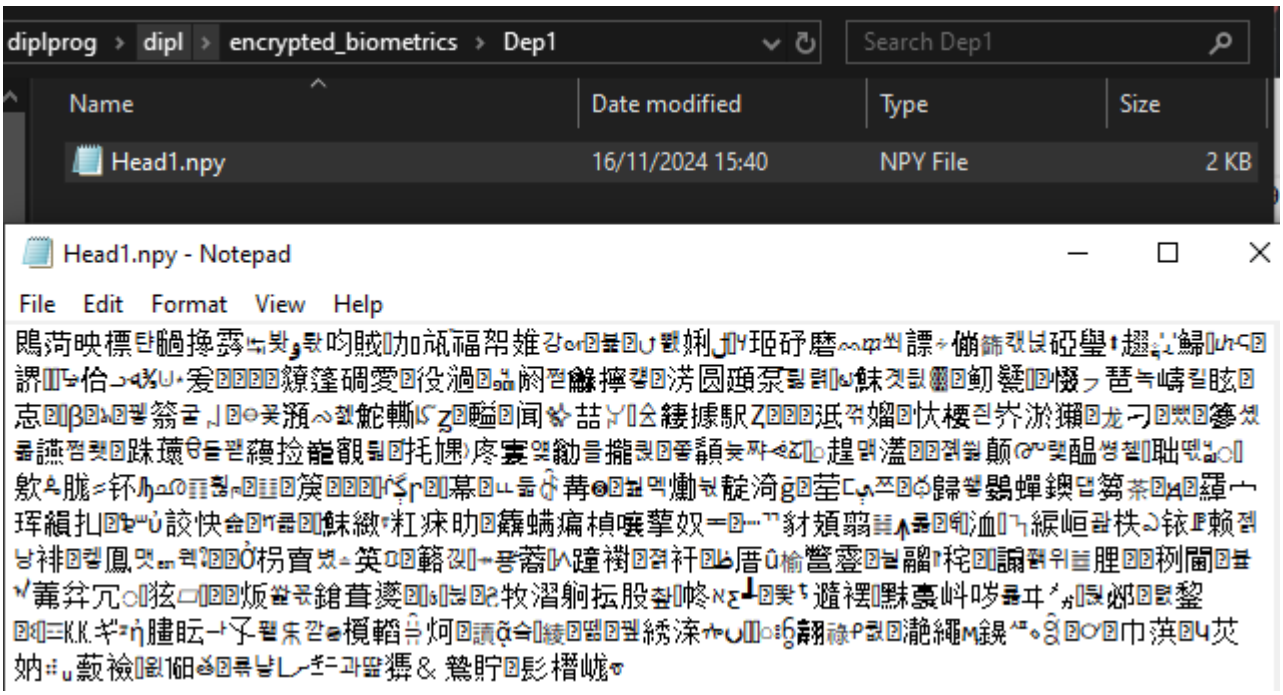

Рис. 3.23. Зашифрований вектор обличчя керівника

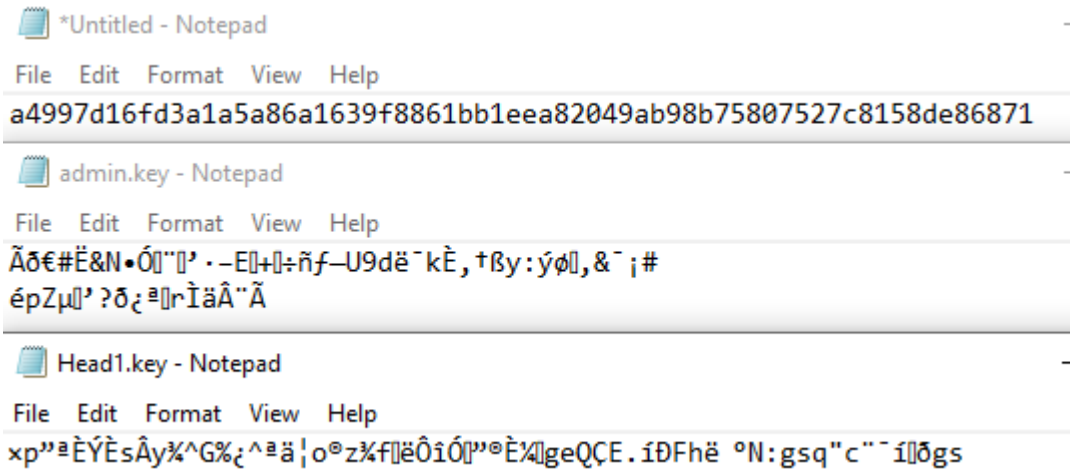

Рис. 3.24. Зашифрований біометричним ключем головний ключ керівника

Тепер продемонструю (на рисунках 3.25-3.30) шифрування та розшифрування файлу за цього ж керівника:

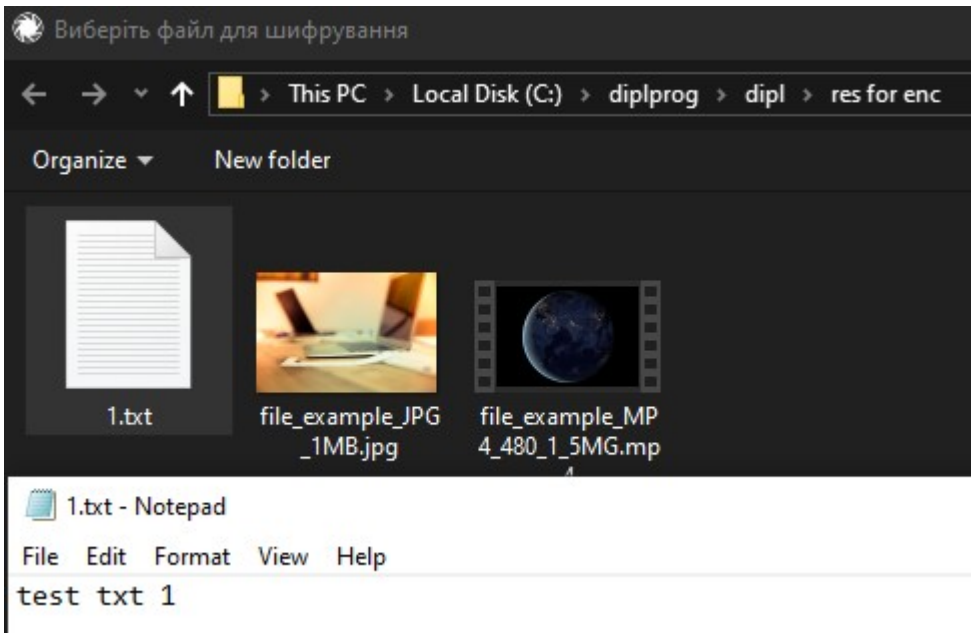

Рис. 3.25. За допомогою діалогового вікна обираю текстовий файл на шифрування

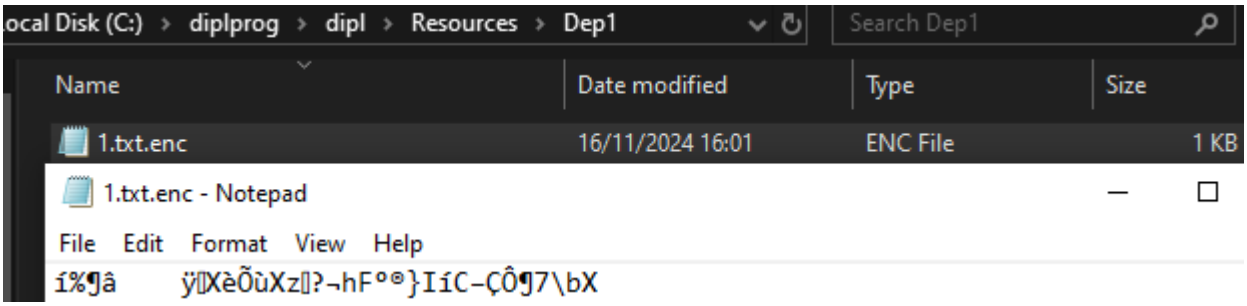

Рис. 3.26. Зашифрований текстовий файл в щойно створеній папці департаменту

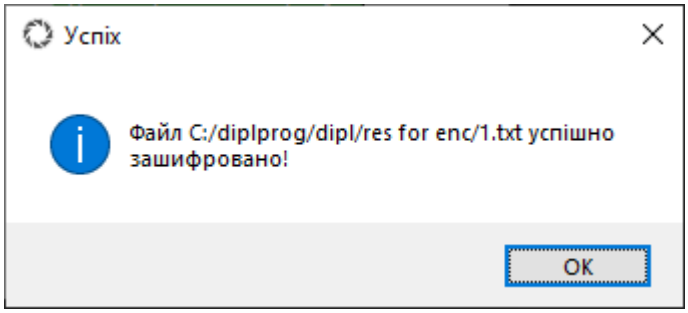

Рис. 3.27. Повідомлення про успішне шифрування файлу

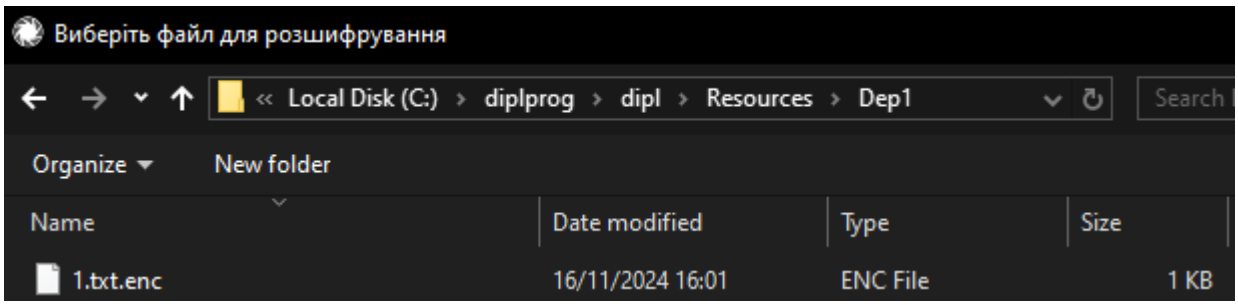

Рис. 3.28. За допомогою діалогового вікна обираю текстовий файл на розшифрування

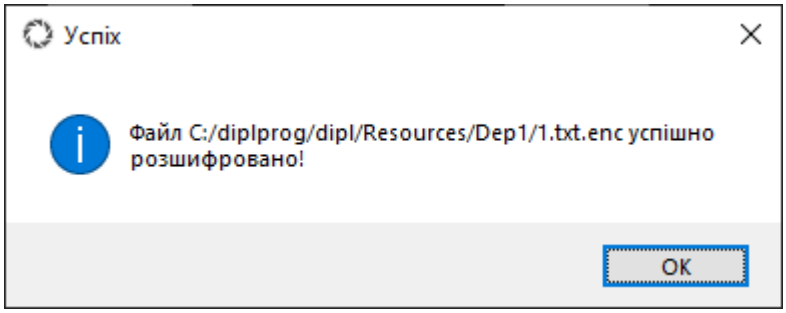

Рис. 3.29. Повідомлення про успішне розшифрування файлу

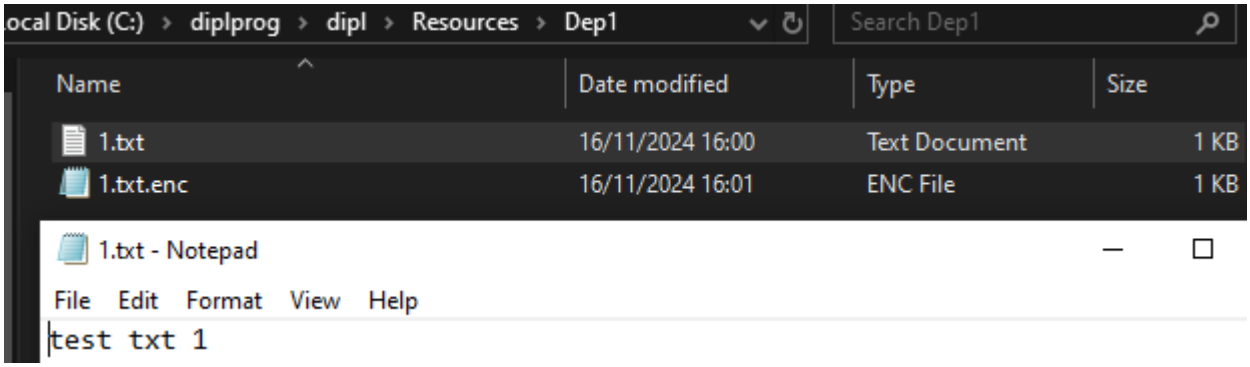

Рис. 3.30. Як бачимо, файл успішно розшифрувався до відповідної папки

Тепер змінимо керівника існуючого департаменту та додамо ще одного (рисунки 2.31 та 2.32), спробуємо розшифрувати (на рисунках 2.33 та 2.34) вже зашифрований текстовий файл. У нового керівника першого департаменту, ключ повинен надатися той самий, тому у нього повинен бути успіх, у керівника нового департаменту – невдача:

main_users.json - Notepad

File Edit Format View Help


```
[
    {
        "username": "admin",
        "password":
"c7ad44cbad762a5da0a452f9e854fdc1e0e7a52a38015f23f3eab1d80b931dd472634dfac71cd
34ebc35d16ab7fb8a90c81f975113d6c7538dc69dd8de9077ec",
        "role": "Administrator"
    },
    {
        "username": "Head2",
        "password":
"3c9909afec25354d551dae21590bb26e38d53f2173b8d3dc3eee4c047e7ab1c1eb8b85103e3be
7ba613b31bb5c9c36214dc9f14a42fd7a2fdb84856bca5c44c2",
        "role": "Manager",
        "department": "Dep1"
    },
    {
        "username": "Head3",
        "password":
"d404559f602eab6fd602ac7680dacbfaadd13630335e951f097af3900e9de176b6db28512f2e0
00b9d04fba5133e8b1c6e8df59db3a8ab9d60be4b97cc9e81db",
        "role": "Manager",
        "department": "Dep2"
    }
]
```


Рис. 3.31. Підтвердження додавання та зміни керівників

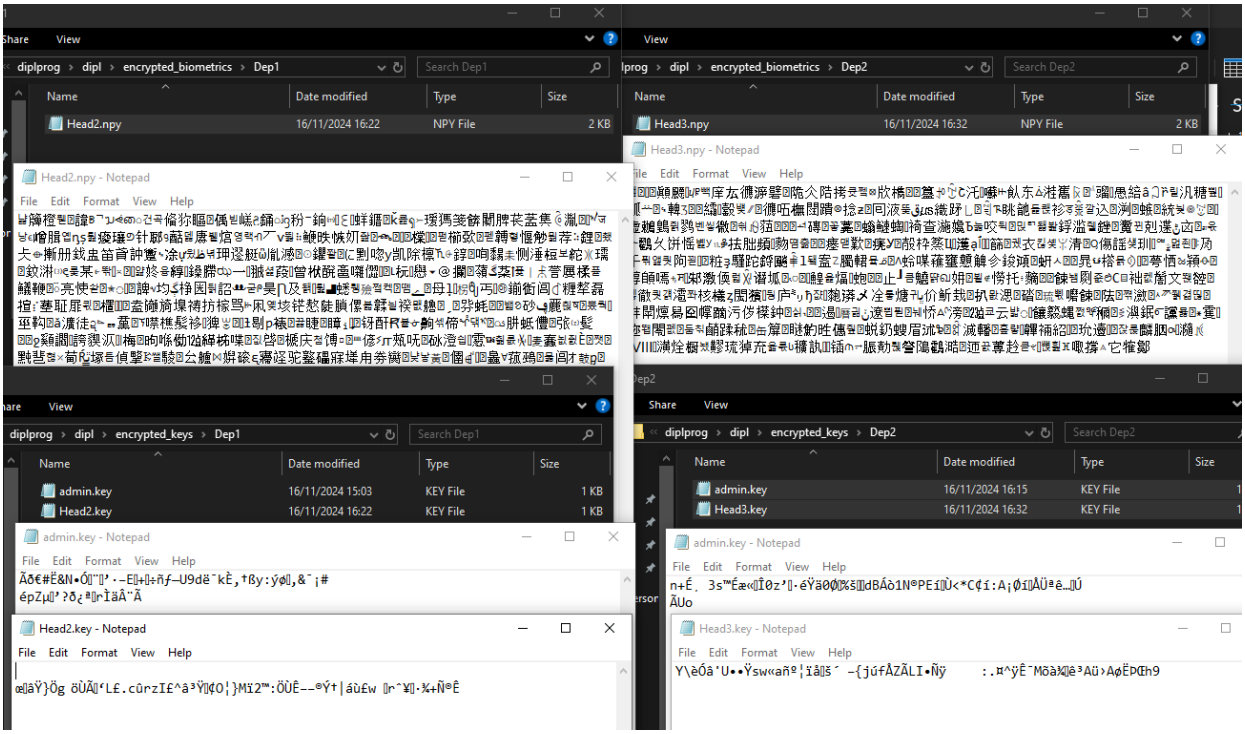

Рис. 3.32. Зашифровані головні ключі та біометрія керівників

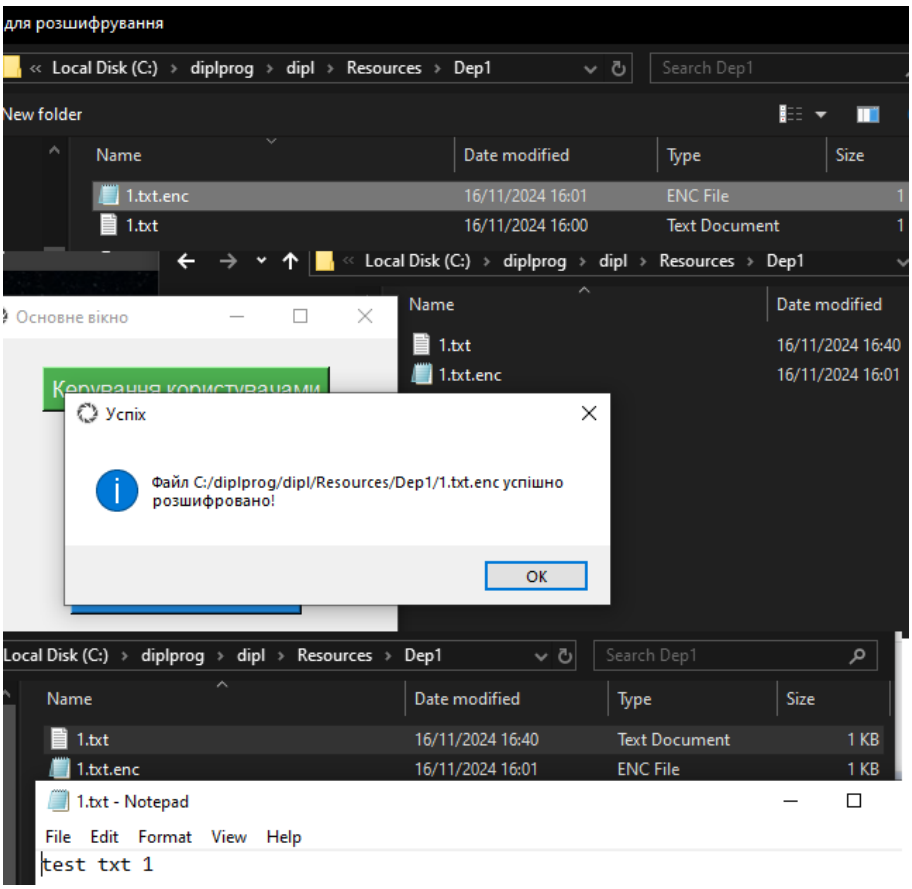

Рис. 3.33. Файл успішно розшифрувався до відповідної папки, як і повинно бути

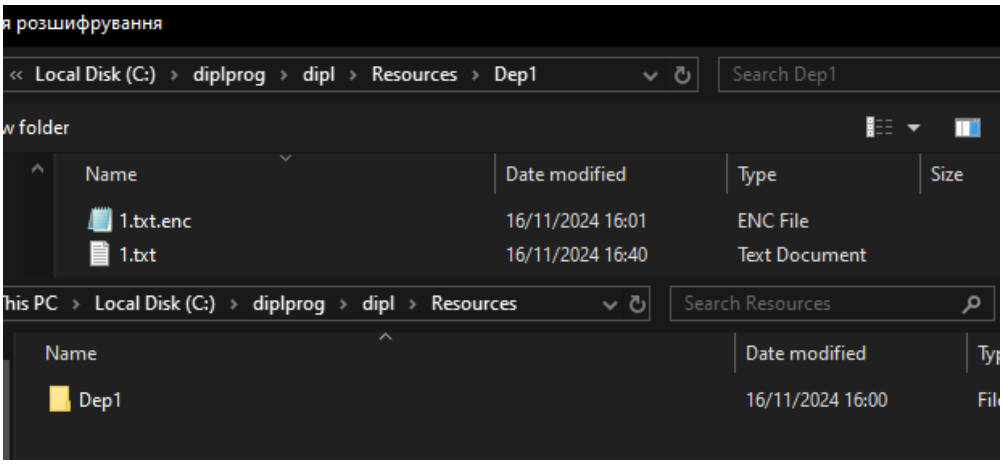

Рис. 3.34. Файл не розшифрувався до папки другого підрозділу, як і повинно бути

Тепер видалимо другого керівника першого підрозділу (на рис 3.36 та 3.37), зашифруємо файл, додамо звичайного користувача до другого підрозділу (на рис. 3.38 та 3.39) та спробуємо розшифрувати файл з під його облікового

запису (на рис. 3.40). Також проведемо зміну пароля адміністратора (на рис. 3.35 та 3.37):

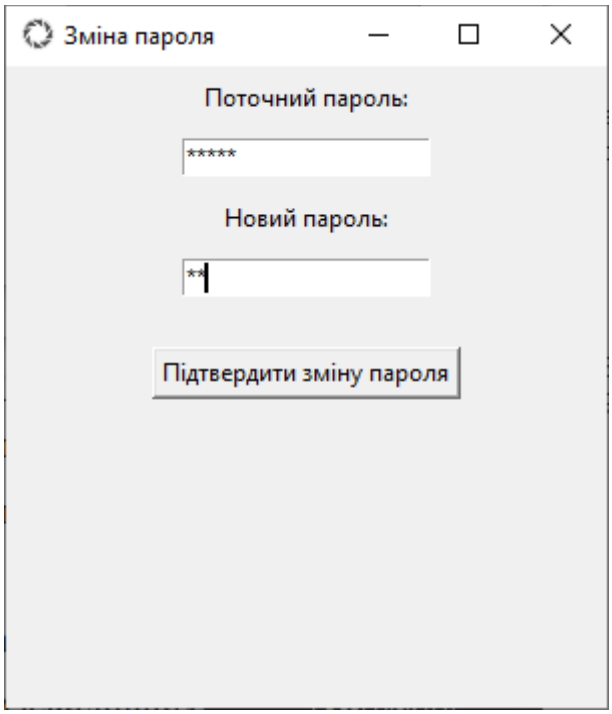

Рис. 3.35. Для зміни паролю вводимо поточний та новий, натискаємо на відповідну кнопку

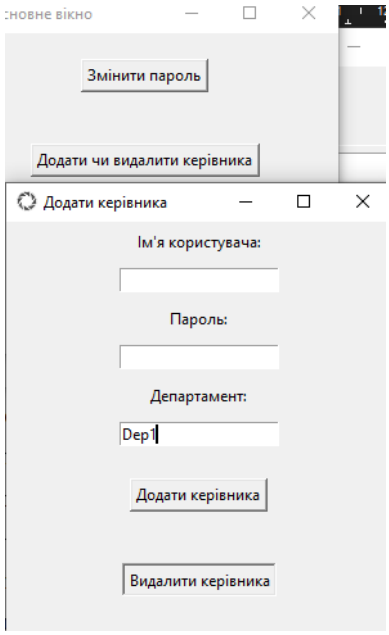

Рис. 3.36. Для видалення керівника вводимо його підрозділ та натискаємо на відповідну кнопку

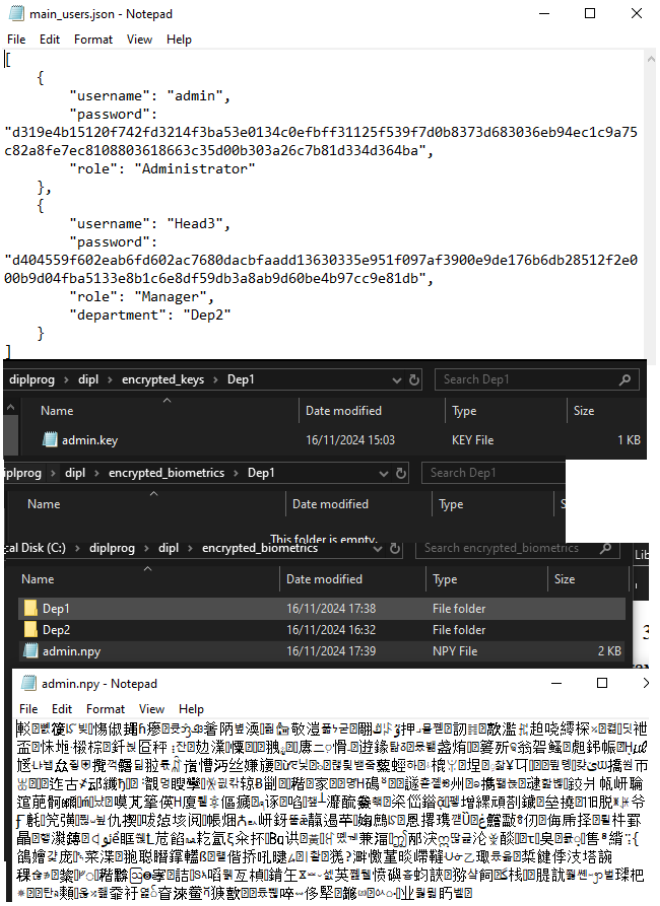

Рис. 3.37. Підтвердження зміни паролю адміністратора та видалення першого керівника

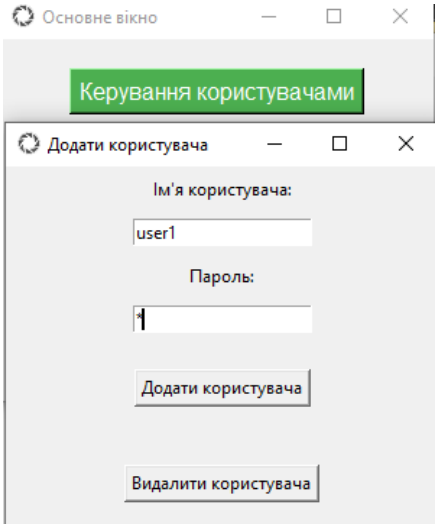

Рис. 3.38. Вікно управління звичайними користувачами

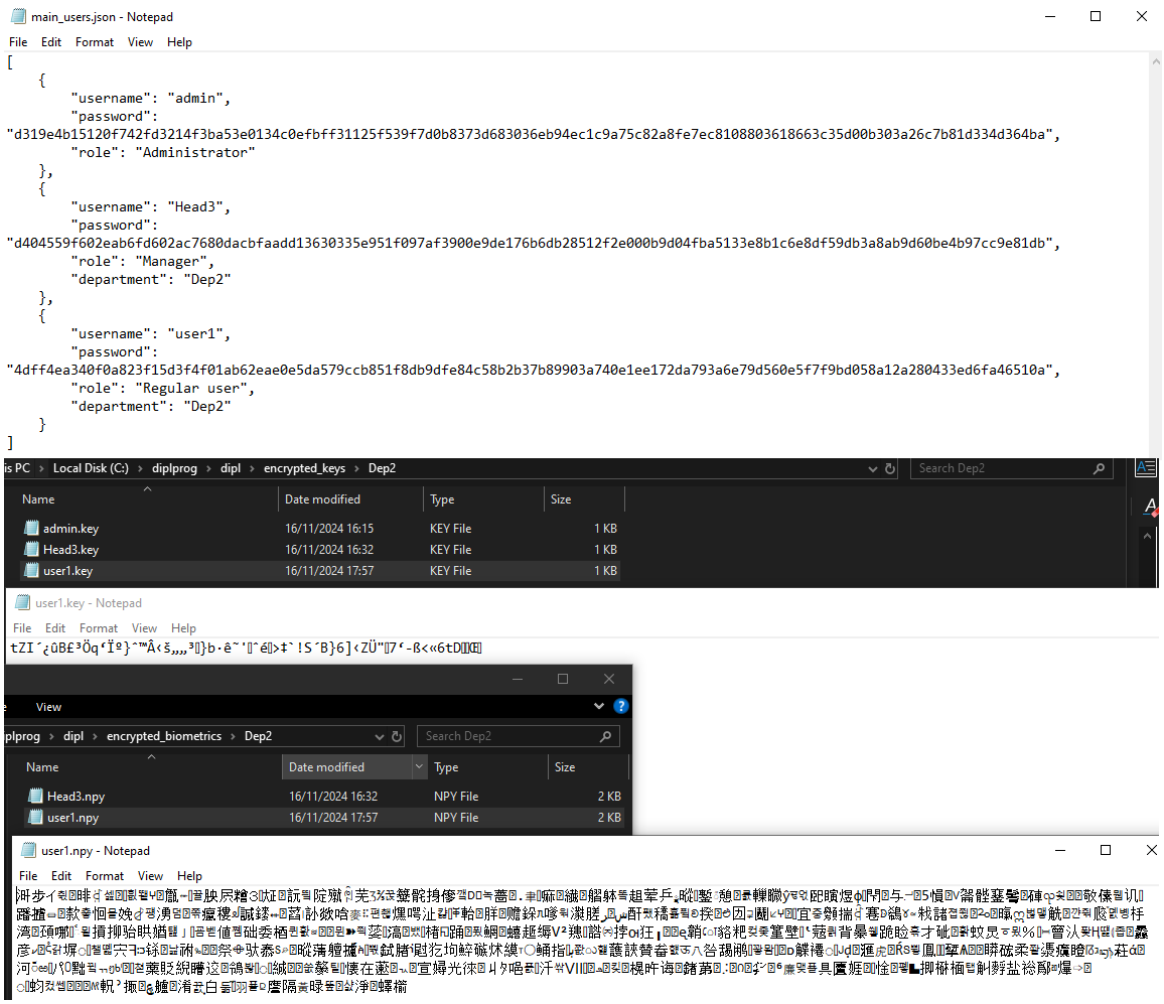

Рис. 3.39. Підтвердження додавання звичайного користувача

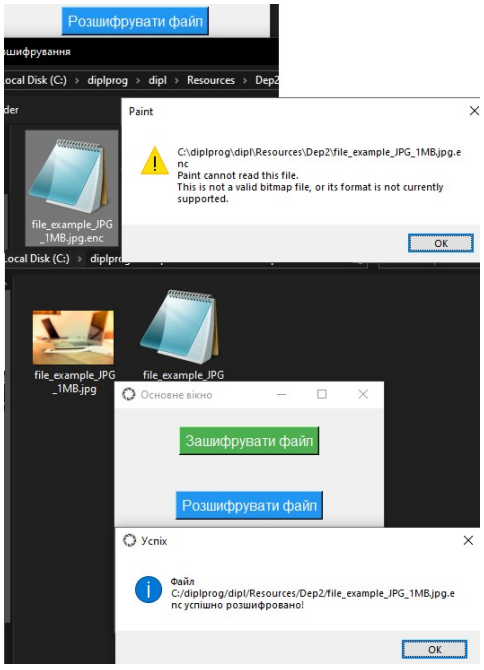

Рис. 3.40. Підтвердження можливості розшифрування

3.3.2 Блок-схеми застосунку (на рисунках 3.41-3.43):

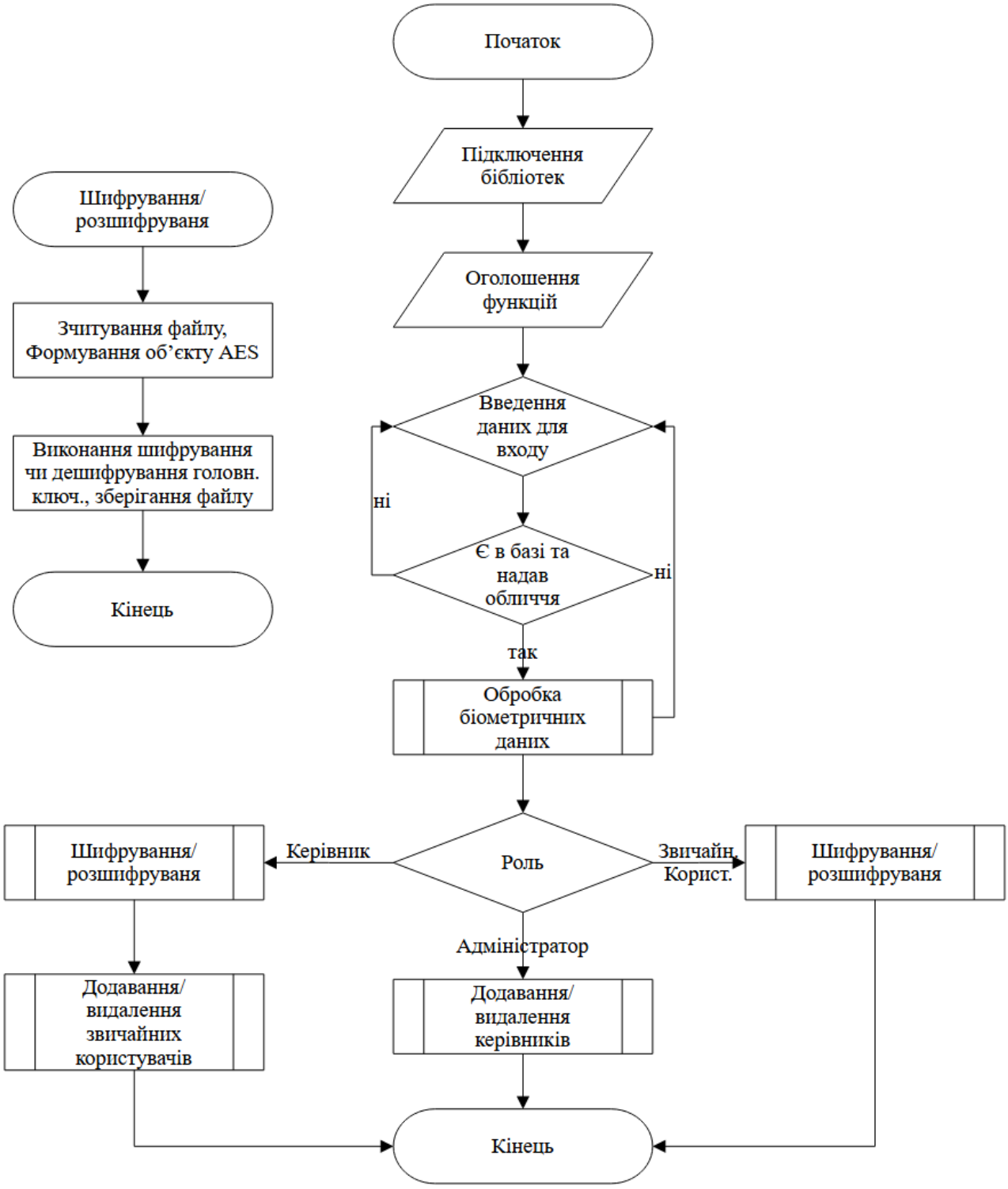

Рис. 3.41. Блок-схема основної частини та підпрограми шифрування

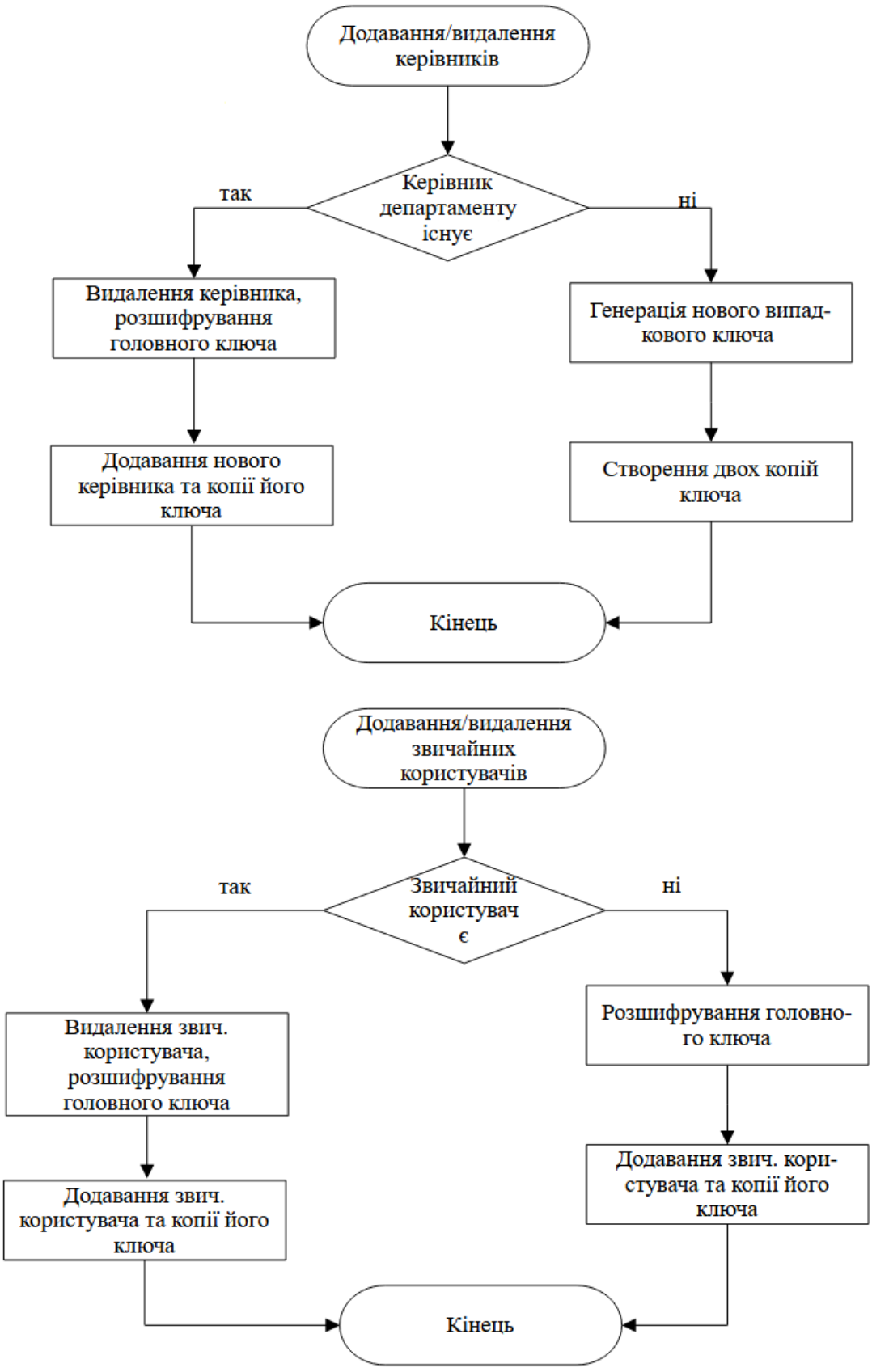

Рис. 3.42. Блок-схема підпрограм додавання користувачів

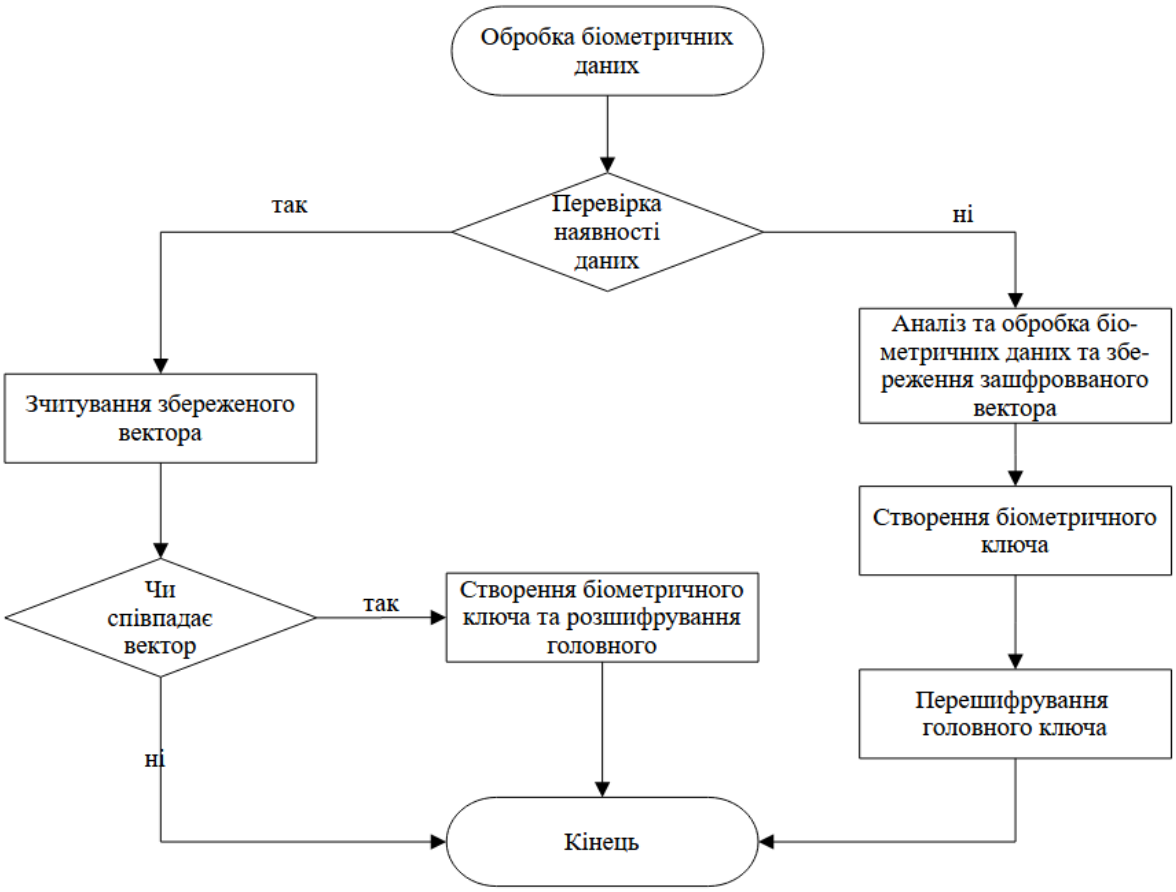

Рис. 3.43. Блок-схема підпрограми обробки біометрії

3.4 Оцінка ефективності програмного модулю

3.4.1 Помилки помилки першого та другого роду

Протестував розроблене програмне рішення на помилки першого та другого роду.

Для тестування на помилки першого роду, тобто відкидання істини використовував десять зображень обличчя однієї людини з різними ракурсами, освітленням та віком. Перед тестуванням я видалив всі результати роботи програми (базу користувачів, ключі, тощо) та створив нового адміністратора, його біометричні дані були взяті з 1.jpg.

Таблиця 3.1

Таблиця тестування на помилки першого роду

Назва зображення	Результат
1.jpg	Успіх
2.jpg	Успіх
3.jpg	Успіх
4.jpg	Успіх
5.jpg	Успіх
6.jpg	Успіх
7.jpg	Успіх
8.jpg	Успіх
9.jpg	Успіх
10.jpg	Успіх

Висновок до таблиці 3.1: оскільки всі тести пройшли успішно, можна зробити висновок, що поріг збігу облич (який визначений у функції def compare_faces() та менше 0.6) був адекватним для розпізнавання варіацій одного обличчя, навіть із різними дуже різними умовами.

Для тестування на помилки другого роду, тобто відкидання істини використовував десять зображень облич різних людей з різними ракурсами, освітленням, віком та статтю. Вони були як і кардинально різні, так і дуже схожі. В якості користувача було використано того самого адміністратора, що має біометричні дані згідно фото 1.jpg.

Таблиця 3.2

Таблиця тестування на помилки другого роду

Назва зображення	Результат
2.1.jpg	Невдача
2.2.jpg	Невдача
2.3.jpg	Невдача
2.4.jpg	Невдача
2.5.jpg	Невдача
2.6.jpg	Невдача
2.7.jpg	Не вдалося розпізнати обличчя
2.8.jpg	Невдача
2.9.jpg	Невдача
2.10.jpg	Невдача

Висновок до таблиці 3.2: У цьому випадку всі зображення, які не належать реальному користувачеві, дали невдачу в порівнянні, крім зображення 2.7.jpg, в ньому не вдалося розпізнати обличчя. Це свідчить про правильну роботу системи з точки зору запобігання несанкціонованого доступу. Щодо невдачі розпізнання обличчя (зображення 2.7.jpg): Це може свідчити про проблему з якістю зображення (наприклад, низька роздільна здатність, або, скоріше за все, погане освітлення, через яке контур обличчя дуже розмитий), що ускладнює правильне розпізнавання

3.4.2 Порівняння з існуючими на ринку аналогами

Таблиця 3.2

Порівняння реалізованого застосунку шифрування ресурсів з аналогами

Характеристика	ESET Endpoint Encryption	Реалізований застосунок	BitLocker	FileVault 2
1	2	3	4	5
Шифрування	AES-256, 3DES, або Blowfish	AES-CBC-256	AES-128 або AES-256 (CBC або XTS)	AES-XTS-128
Кросплатформність	Windows 10, 8, 8.1, 7, Vista, XP, Server 2003 - Server 2016, macOS Mojave та новіші, iOS	Windows 10 та новіші, повинна працювати на актуальних версіях Linux та macOS, але не тестувався на них	Від Enterprise та Ultimate версії Windows Vista до Pro, Enterprise, та Education версії Windows 11	macOS X 10.7 Lion та новіші
Підтримка TPM	Підтримує	Не підтримує	Підтримує	Не підтримує
Режими шифрування	Повне шифрування диска, файлів, папок і змінних носіїв. Підтримка шифрування електронної пошти, тексту і буфера обміну	Шифрування окремих файлів	Повне шифрування диска	Повне шифрування диска
Особливості	Дуже багатий функціонал	Інтеграція системи розпізнавання облич	Є частиною операційної системи	Є частиною операційної системи

Закінчення таблиці 3.2

1	2	3	4	5
Управління ключами	Централізоване та локальне управління ключами	Локальне управління ключами	Локальне управління ключами	Централізоване та локальне управління ключами з можливістю зберігання ключів відновлення на серверах Apple
Простота використання	Гнучке налаштування для підприємств, централізоване управління	Простий, мінімалістичний інтерфейс, мінімальна кількість налаштувань	Інтегроване у Windows, мінімальна кількість налаштувань	Інтегроване у macOS, мінімальна кількість налаштувань

Висновки до таблиці:

ESET Endpoint Encryption має найбільш гнучкий функціонал та підтримку різних платформ алгоритмів шифрування. Особливо підходить для великих підприємств і організацій, де потрібне централізоване управління безпекою та гнучкість у виборі методів шифрування.

Реалізований застосунок забезпечує високий рівень безпеки, за рахунок використання AES-CBC-256 та інтегрування біометрії, що робить його більш зручним для користувачів, які шукають простоту і додаткову безпеку водночас, але має дещо обмежену кросплатформність, лише локальне управління ключами.

BitLocker і FileVault 2 є простими в використанні інструментами для шифрування диска, найменшу гнучкість і обмежену підтримку платформ.

3.5 Висновки до третього розділу

У даному розділі були визначені вимоги до функціоналу застосунку, до мови програмування, що повинна використовуватися для практичної реалізації.

В результаті обрав мову Python, під мій випадок вона підходила найбільше.

Окрім мови були розглянуті використовувані бібліотеки.

Була проведена розробка програмного застосунку для шифрування ресурсів з використанням системи розпізнання облич. Де детально описав основні функції, як вони працюють.

Також особливу увагу приділив тому, як перетворюється фото обличчя на біометричний ключ, коли та для чого він використовується. Як генеруються інші два типи ключів, з паролів та криптографічно стійким генератором псевдовипадкових чисел. Як працює передача ключів.

Також не оминув увагою і саме шифрування, звичайно, яке реалізовано використовуючи симетричний алгоритм шифрування AES.

Після цього було проведено детальне тестування всіх функцій програми.

Останніми пунктами було проведено порівняння з існуючими на цьому ринку аналогами, тестування на помилки першого та другого роду, що є справжньою проблемою в біометричних системах. Буває так, що вони то не пропускають тих, хто має право на пропуск, або ще чого гірше, приймають злочинців за таких. Але під час проведення даного тестування застосунок показав себе з гарного боку.

ВИСНОВКИ

В умовах швидкого розвитку технологій та постійно зростаючої кількості кіберзагроз, традиційні системи захисту інформації часто не здатні забезпечити належний рівень безпеки, що створює ризики витоку даних або їх несанкціонованого доступу. Саме тому під час виконання даної кваліфікаційної роботи було проведено дослідження сучасних підходів до забезпечення інформаційної безпеки та був розроблений програмний застосунок, який для захисту даних від несанкціонованого доступу, поєднує біометричні технології з класичними методами шифрування.

В рамках дослідження було проведено глибокий аналіз сучасного стану забезпечення інформаційної безпеки. Розглянуто існуючі загрози в інформаційних мережах, особливості атак на системи, а також способи їхньої реалізації. Детально проаналізовано традиційні алгоритми шифрування, серед яких окрему увагу приділено симетричному алгоритму Advanced Encryption Standard, який і використовується як основний в розробленому застосунку. Також у ході роботи було проаналізовано існуючі програмні рішення для шифрування ресурсів, що дозволило визначити їх переваги та недоліки й виокремити можливості для вдосконалення, що й було реалізовано в рамках цієї роботи. Приділено увагу біометрії, яка використовуються для ідентифікації користувачів, зокрема методам розпізнавання облич. Завдяки унікальності геометричних характеристик людського обличчя та високій точності сучасних алгоритмів розпізнавання, застосування таких методів суттєво підвищує рівень захисту та ускладнює спроби крадіжки ресурсів.

Під час розробки було створено застосунок, який складається з кількох модулів, де головними є модулі біометричний та шифрування. Кожен із них виконує чітко визначену функцію та забезпечує високий рівень безпеки системи в цілому. Модуль шифрування реалізує шифрування даних за допомогою алгоритму AES, цей симетричний алгоритм шифрування забезпечує надійний

захист завдяки використанню складних криптографічних механізмів, але при цьому і працює швидко. Модуль керування ключами дозволяє генерувати, зберігати та передавати криптографічні ключі, забезпечуючи можливість багаторівневого захисту. Біометричний модуль використовує систему розпізнавання облич для підтвердження особи користувача, що робить систему більш стійкою до несанкціонованого доступу, оскільки обличчя, як унікальний ідентифікатор, неможливо втратити або передати іншим особам, як це може статися з паролями. Було інтегровано систему розпізнавання облич із класичними методами шифрування. Зокрема, процес шифрування включає використання ключа, який генерується на основі біометричних характеристик. Це дозволяє мінімізувати ризики, пов'язані з втратою апаратних ключів, і водночас забезпечує зручність у використанні системи. Усі модулі застосунку працюють у тісній взаємодії, утворюючи єдиний комплекс, який поєднує традиційні методи шифрування з сучасними біометричними технологіями.

Розроблений застосунок було протестовано з метою перевірки його ефективності, надійності та зручності використання. Результати тестування показали, що застосунок здатний забезпечити високий рівень захисту інформації, залишаючись при цьому інтуїтивно зрозумілим і зручним для користувачів. Порівняльний аналіз із існуючими на ринку аналогами підтвердив наявність переваг розробленого застосунку, серед яких використання біометричних даних є основною перевагою.

Таким чином, можна зробити висновок, що запропонований метод та його реалізація мають значний потенціал для практичного застосування у сфері забезпечення інформаційної безпеки, як для персонального використання, так і для корпоративних потреб. Результати роботи можуть стати основою для подальших досліджень у напрямі вдосконалення систем захисту інформації та впровадження нових технологій у цю сферу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Big Breaches: What we learned from some of the world's most disruptive cyberattacks. URL: https://online.stanford.edu/big-breaches-what-we-learned-some-worlds-most-disruptive-cyberattacks?trk=article-ssr-frontend-pulse_little-text-block. (дата звернення: 12.09.2024)
2. Privacy and Data Protection by Design – from policy to engineering. URL: https://www.enisa.europa.eu/publications/privacy-and-data-protection-by-design/@@download/fullReport. (дата звернення: 12.09.2024)
3. Regulation (EU) 2016/679 of the European Parliament and of the Council. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj#d1e3063-1-1. (дата звернення: 12.09.2024)
4. PCI DSS Quick Reference Guide: Understanding the Payment Card Industry Data Security Standard version 3.2.1. URL: https://listings.pcisecuritystandards.org/documents/PCI_DSS-QRG-v3_2_1.pdf. (дата звернення: 12.09.2024)
5. Richard A. Mollin. An INTRODUCTION to CRYPTOGRAPHY. Boca Raton: Taylor & Francis Group, 2007. 86 с.
6. Richard A. Mollin. An INTRODUCTION to CRYPTOGRAPHY. Boca Raton: Taylor & Francis Group, 2007. 152 с.
7. Richard A. Mollin. An INTRODUCTION to CRYPTOGRAPHY. Boca Raton: Taylor & Francis Group, 2007. 162 с.
8. Linus Gasser, Mulder, Valentin; Mermoud, Alain; Lenders, Vincent; Tellenbach, Bernhard (eds.), "Post-quantum Cryptography", Trends in Data Protection and Encryption Technologies, Cham: Springer Nature Switzerland, 2023. pp. 47-52
9. Xavier Bonnetain, María Naya-Plasencia, André Schrottenloher. Quantum Security Analysis of AES. IACR Transactions on Symmetric Cryptology, 2019, 2019 (2), pp.55-93

10. Introduction (SSL/TLS in Windows Server 2003). URL: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc757054(v=ws.10)#differences-between-ssl-and-tls. (дата звернення: 12.09.2024)

11. TLS Session Resumption: Full-speed and Secure. URL: https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure. (дата звернення: 12.09.2024)

12. An Illustrated Guide to IPsec. URL: ipsec https://unixwiz.net/techtips/iguide-ipsec.html (дата звернення: 12.09.2024)

13. RFC 9580 OpenPGP. URL: https://datatracker.ietf.org/doc/html/rfc9580 (дата звернення: 12.09.2024)

14. Achary R. Cryptography And Networking Security: An Introduction. Dulles: Mercury Learning and Information, 2021. - 440 c.

15. Marius Iulian Mihailescu, Stefania Loredana Nita. Pro Cryptography and Cryptanalysis with C++20. New York: Apress Media, 2021. 259-260 c.

16. ESET Buys Recognized Data Encryption Leader DESlock. URL: https://www.eset.com/int/about/newsroom/press-releases/products/eset-buys-recognized-data-encryption-leader-deslock. (дата звернення: 12.09.2024)

17. Вебсайт VeraCrypt. URL: https://www.veracrypt.fr/en/Home.html. (дата звернення: 10.09.2024)

18. Стаття про BitLocker в бібліотеці технічної документації Microsoft Learn. URL: https://learn.microsoft.com/en-us/windows/security/operating-system-security/data-protection/bitlocker. (дата звернення: 10.09.2024)

19. Apple Technical White Paper – Best Practices for Deploying FileVault 2. URL: https://kryptera.se/assets/uploads/2014/10/WP_FileVault2.pdf. (дата звернення: 10.09.2024)

20. Alenezi, M. N., Alabdulrazzaq, H. K., & Mohammad, N. Q. (2022). Symmetric Encryption Algorithms: Review and Evaluation Study. International

Journal of Communication Networks and Information Security (IJCNIS), 12(2). 2020. p. 272. DOI:10.17762/ijcnis.v12i2.4698

21. Advanced Encryption Standard (AES). Federal Information Processing Standards. Gaithersburg, MD. 2001. p. 12. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf

22. Bertoni, Guido; Breveglieri, Luca; Fragneto, Pasqualina; MacChetti, Marco; Marchesin, Stefano (2003). "Efficient Software Implementation of AES on 32-Bit Platforms". Cryptographic Hardware and Embedded Systems - CHES 2002. Lecture Notes in Computer Science. Vol. 2523. pp. 159–171. doi:10.1007/3-540-36400-5_13. ISBN 978-3-540-00409-7.

23. Bruce Schneier. "Another New AES Attack". Schneier on Security, A blog covering security and security technology. URL: https://www.schneier.com/blog/archives/2009/07/another_new_aes.html (дата звернення: 10.11.2024)

24. Biaoshuai Tao & Hongjun Wu. "Improving the Biclique Cryptanalysis of AES". Information Security and Privacy. Lecture Notes in Computer Science. Vol. 9144. 2015. pp. 39–56. doi:10.1007/978-3-319-19962-7_3. ISBN 978-3-319-19961-0.

25. Michael Neve and Kris Tiri. On the complexity of side-channel attacks on AES-256. Intel Corporation, JF5-254. 2007. URL: https://eprint.iacr.org/2007/318.pdf (дата звернення: 10.11.2024)

26. Richard Evers, Alastair Sweeny PhD. Quantum computers will not cause all forms of conventional encryption to become insecure. 2019. URL: https://kryptera.ca/paper/2019-03/ (дата звернення: 10.11.2024)

27. Paul Smith-Goodson Contributor Moor Insights and Strategy. Atom Computing Announces Record-Breaking 1,225-Qubit Quantum Computer. 2023. URL:https://www.forbes.com/sites/moorinsights/2023/10/24/atom-computing-announces-record-breaking-1225-qubit-quantum-computer/ (дата звернення: 10.11.2024)

28. Test results for amd64, hertz, crypto_hash. 2024. URL:bench.cr.yp.to/web-impl/amd64-hertz-crypto_hash.html (дата звернення: 10.11.2024)

29. Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini. The First Collision for Full SHA-1. 2017. DOI:10.1007/978-3-319-63688-7_19

30. Tao Xie, Fanbao Liu, Dengguo Feng. Fast Collision Attack on MD5. URL: https://eprint.iacr.org/2013/170.pdf (дата звернення: 10.11.2024)

31. Secure Hash Standard (SHS). Federal Information Processing Standards. Gaithersburg, MD. 2015. pp. 21-24. DOI:10.6028/NIST.FIPS.180-4

32. Yingxin Li, Fukang Liu, Gaoli Wang. New Records in Collision Attacks on SHA-2. 2024. URL: https://eprint.iacr.org/2024/349.pdf (дата звернення: 10.11.2024)

33. Jean-Philippe Aumasson. Serious Cryptography A Practical Introduction to Modern Encryption. No Starch Press. San Francisco. 2017. pp. 22-37

34. Cryptography API: Next Generation. URL: https://learn.microsoft.com/en-us/windows/win32/seccng/cng-portal (дата звернення: 10.11.2024)

35. Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. INRIA Rhone-Alps. France. URL: https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf (дата звернення: 11.11.2024)

36. Vahid Kazemi and Josephine Sullivan. One Millisecond Face Alignment with an Ensemble of Regression Trees. KTH, Royal Institute of Technology. Sweden. 2014. URL: https://www.csc.kth.se/~vahidk/papers/KazemiCVPR14.pdf

37. Dan C. Cires¸an, Ueli Meier, Jonathan Masci, Luca M. Gambardella, Jurgen Schmidhuber. Flexible, High Performance Convolutional Neural Networks for Image Classification. Switzerland. 2011. DOI:10.5591/978-1-57735-516-8/IJCAI11-210

38. Why was Python created in the first place?. URL: https://docs.python.org/3/faq/general.html#id7 (дата звернення: 11.11.2024)

39. Niklas Heer. Speed comparison of programming languages. URL: https://github.com/niklas-heer/speed-comparison (дата звернення: 11.11.2024)

40. Adam Geitgey. Face Recognition. URL: https://github.com/ageitgey/face_recognition?tab=readme-ov-file (дата звернення: 11.11.2024)

41. Open Source Computer Vision Library. URL: https://github.com/opencv/opencv (дата звернення: 11.11.2024)

ДОДАТКИ

Додаток А

Код програмного застосунку

```
import json
import os
import tkinter as tk
from tkinter import messagebox, filedialog
import cv2
import face_recognition
import numpy as np
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
from Crypto.Hash import SHA512, SHA256



# Шлях до файлу з даними користувачів
DATA_FILE = 'main_users.json'

#Папка для збереження біометричних даних і паролів
encrypted_bio_folder = "encrypted_biometrics"
encrypted_key_folder = "encrypted_keys"
def read_user_data():
    """Зчитує дані користувачів з файлу."""
    if not os.path.exists(DATA_FILE):
        # Якщо файл не існує, створюємо його з одним стандартним користувачем
        default_users = [{'username': 'admin', 'password': SHA512.new("admin".encode()).hexdigest(),
'role': 'Administrator'}]
        write_user_data(default_users)  # Записуємо стандартного користувача в файл
        return default_users
```

```
    with open(DATA_FILE, 'r', encoding='utf-8') as file:
        try:
            return json.load(file)  # Зчитуємо дані з JSON-файлу
        except json.JSONDecodeError:
            return False

def write_user_data(users):
    """Записує дані користувачів у файл."""
    with open(DATA_FILE, 'w', encoding='utf-8') as file:
        json.dump(users, file, indent=4)  # Записуємо дані

def login():
    global encrypt_window
    if encrypt_window is not None and encrypt_window.winfo_exists():
        encrypt_window.destroy()  # Якщо вікно вже відкрите
    """Отримує дані користувача за логіном."""
    inputted_name = username_entry.get()
    password = password_entry.get()

    def set_placeholder(entry, placeholder):
        entry.delete(0, 'end')  # Очистити поле
        entry.insert(0, placeholder)  # Вставити плейсхолдер
    set_placeholder(username_entry, "Ваш логін")
    set_placeholder(password_entry, "Ваш пароль")


    username_entry.bind("<FocusIn>", lambda e: clear_placeholder(username_entry, "Ваш логін"))
    password_entry.bind("<FocusOut>", lambda e: restore_placeholder(password_entry, "Ваш
пароль"))
    hashed_password = SHA512.new(password.encode()).hexdigest()
    password_key = SHA256.new(password.encode()).digest()    # Створюємо ключ з пароля
    users = read_user_data()
    for user in users:
        if user['username'] == inputted_name :
```

```
            if user['password'] == hashed_password:
                user_id = user['username']
                print(f"{user['username']} logged in")
                if user['role']!="Administrator":
                    global encrypted_key_folder
                    encrypted_key_folder = f"encrypted_keys/{user['department']}"
                    global encrypted_bio_folder
                    encrypted_bio_folder = f"encrypted_biometrics/{user['department']}"
                else:
                    encrypted_key_folder = "encrypted_keys"
                    encrypted_bio_folder = "encrypted_biometrics"
                if os.path.exists(f"{encrypted_bio_folder}/{user_id}.npy"):
                    # Завантажуємо біометричні дані
                    saved_face_encoding = load_encrypted_biometric_data(user_id, password_key)
                    image = capture_face_image()
                    new_face_encoding = get_face_encoding(image)

                    if not compare_faces(saved_face_encoding, new_face_encoding):
                        print("Обличчя не збігається.")
                        return
                    open_encrypt_window(user, password_key)  # Відкрити нове вікно після успішного
входу
                else:
                    # Захоплення та збереження нових біометричних даних
                    image = capture_face_image()
                    face_encoding = get_face_encoding(image)
                    if user['role']!="Administrator":
                        with open(f"{encrypted_key_folder}/{user_id}.key", "rb") as f:
                            iv = f.read(16)  # IV для CBC режиму
                            pass_encrypted_key = f.read()
                            # Розшифровуємо ключ (пароль)
                        cipher = AES.new(password_key, AES.MODE_CBC, iv)
                        dep_key = unpad(cipher.decrypt(pass_encrypted_key), AES.block_size)
```

```
                encrypted_dep_key = encrypt_master_key_with_face(dep_key, face_encoding)
                with open(f"{encrypted_key_folder}/{user['username']}.key", "wb") as f:
                    f.write(encrypted_dep_key)

            # Шифрування біометричних даних за допомогою пароля
            save_encrypted_biometric_data(user_id, password_key, face_encoding)
            open_encrypt_window(user, password_key)  # Відкрити нове вікно після успішного
входу
        return user
    print("no s such user")
    return None



"""Face thing"""

#    Шифрує та зберігає вектор ознак обличчя, використовуючи пароль
def save_encrypted_biometric_data(user_id, password_key, face_encoding):
    if not os.path.exists(encrypted_bio_folder):
        os.makedirs(encrypted_bio_folder)



    # Шифруємо вектор ознак обличчя
    cipher = AES.new(password_key, AES.MODE_CBC)
    encrypted_face_encoding = cipher.encrypt(pad(face_encoding.tobytes(), AES.block_size))
    # Зберігаємо зашифрований вектор ознак обличчя і IV
    with open(f"{encrypted_bio_folder}/{user_id}.npy", "wb") as f:
        f.write(cipher.iv + encrypted_face_encoding)

#    Завантажує та розшифровує вектор ознак обличчя, використовуючи пароль
def load_encrypted_biometric_data(user_id, password_key):

    # Завантажуємо зашифрований вектор і IV
    with open(f"{encrypted_bio_folder}/{user_id}.npy", "rb") as f:
```

```
            iv = f.read(16)  # IV для CBC режиму
            encrypted_face_encoding = f.read()

        # Розшифровуємо вектор ознак обличчя
        cipher = AES.new(password_key, AES.MODE_CBC, iv)
        decrypted_face_encoding = unpad(cipher.decrypt(encrypted_face_encoding), AES.block_size)

        return np.frombuffer(decrypted_face_encoding, dtype=np.float64)

    #    Шифрує головний AES ключ за допомогою обличчя
    def encrypt_master_key_with_face(master_key, face_encoding):
        face_key = SHA256.new(face_encoding.tobytes()).digest()
        cipher = AES.new(face_key, AES.MODE_CBC)
        encrypted_master_key = cipher.encrypt(pad(master_key, AES.block_size))
        return cipher.iv + encrypted_master_key

    #    Розшифровує головний AES ключ за допомогою обличчя
    def decrypt_master_key_with_face(enc_master_key, face_encoding):
        face_key = SHA256.new(face_encoding.tobytes()).digest()
        iv = enc_master_key[:16]
        encrypted_key = enc_master_key[16:]
        cipher = AES.new(face_key, AES.MODE_CBC, iv)
        return unpad(cipher.decrypt(encrypted_key), AES.block_size)

    #    Шифрує дані за допомогою головного ключа
    def encrypt_data(data, master_key):
        cipher = AES.new(master_key, AES.MODE_CBC)
        encrypted_data = cipher.encrypt(pad(data, AES.block_size))
        return cipher.iv + encrypted_data

    #    Розшифровує дані за допомогою головного ключа
    def decrypt_data(enc_data, master_key):
        iv = enc_data[:16]
        encrypted_data = enc_data[16:]
```

```
    cipher = AES.new(master_key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(encrypted_data), AES.block_size)

#   Захоплення зображення з камери
def capture_face_image():
    video_capture = cv2.VideoCapture(0)
    print("'q' для фото")

    while True:
        ret, frame = video_capture.read()
        if not ret:
            print("Не вдалося захопити відео.")
            break
        cv2.imshow('Video', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    video_capture.release()
    cv2.destroyAllWindows()

    return cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

#   Отримуємо вектор ознак обличчя
def get_face_encoding(image):
    face_locations = face_recognition.face_locations(image)
    face_encodings = face_recognition.face_encodings(image, face_locations, model="large")

    if len(face_encodings) > 0:
        return face_encodings[0]
    else:
        raise Exception("Обличчя не знайдено")

#   Порівнює два вектори ознак обличчя
```

```
def compare_faces(face_encoding_1, face_encoding_2):
    distance = np.linalg.norm(face_encoding_1 - face_encoding_2)
    return distance < 0.6  # Поріг збігу

#    Шифрування файлу
def encrypt_file_wf(filepath, user_id, passwordkey, department, role):
    if role == "Regular user" and os.path.exists(f"Resources\\{department}\\{os.path.split(filepath)
[1]}.enc"):
        messagebox.showinfo("Невдача", f"Заміна існуючих файлів для звичайних користувачів
недоступна", parent=encrypt_window)
        return False

    if not os.path.exists(f"Resources\\{department}"):
        os.makedirs(f"Resources\\{department}")
    # Завантажуємо біометричні дані
    saved_face_encoding = load_encrypted_biometric_data(user_id, passwordkey)

    # Завантажуємо зашифрований головний ключ
    with open(f"{encrypted_key_folder}/{user_id}.key", "rb") as f:
        encrypted_master_key = f.read()

    # Розшифровуємо головний ключ
    master_key = decrypt_master_key_with_face(encrypted_master_key, saved_face_encoding)



    # Читання файлу
    with open(filepath, "rb") as f:
        file_data = f.read()

    # Шифрування даних
    encrypted_data = encrypt_data(file_data, master_key)



    # Збереження зашифрованих даних
```

```
    with open(f"Resources\\{department}\\{os.path.split(filepath)[1]}.enc", "wb") as f:
        f.write(encrypted_data)
    messagebox.showinfo("Успіх", f"Файл {filepath} успішно зашифровано!",
parent=encrypt_window)

#    Розшифрування файлу
def decrypt_file_wf(filepath, user_id, passwordkey):
    # Завантаження біометричних даних
    saved_face_encoding = load_encrypted_biometric_data(user_id, passwordkey)
    # Читання зашифрованого ключа
    with open(f"{encrypted_key_folder}/{user_id}.key", "rb") as f:
        encrypted_master_key = f.read()

    # Розшифрування головного ключа за допомогою обличчя
    master_key = decrypt_master_key_with_face(encrypted_master_key, saved_face_encoding)

    # Читання зашифрованих даних
    with open(filepath, "rb") as f:
        enc_data = f.read()

    # Розшифрування даних
    decrypted_data = decrypt_data(enc_data, master_key)

    # Збереження розшифрованого файлу
    decrypted_filepath = filepath.replace(".enc", "")
    with open(decrypted_filepath, "wb") as f:
        f.write(decrypted_data)
    print(f"Файл розшифровано і збережено як: {decrypted_filepath}")

"""face thing"""

# Функція для основного вікна
encrypt_window = None  # Змінна для збереження вікна менеджера файлів
file_window = None  # Змінна для збереження вікна менеджера файлів
```

```
def open_encrypt_window(user, password_key):
    global encrypt_window
    if encrypt_window is not None and encrypt_window.winfo_exists():
        encrypt_window.destroy()   # Якщо вікно вже відкрите
    # Створення нового вікна
    encrypt_window = tk.Toplevel(root)
    encrypt_window.title("Основне вікно")
    encrypt_window.iconbitmap("yeet.ico")
    encrypt_window.geometry("300x250")

    if user['role']!="Administrator":
        def encrypt_dialog():
            filepath = filedialog.askopenfilename(title="Виберіть файл для шифрування",
parent=encrypt_window)
            if filepath:
                encrypt_file_wf(filepath, user['username'],password_key, user['department'], user['role'])

        def decrypt_dialog():
            filepath = filedialog.askopenfilename(title="Виберіть файл для розшифрування",
parent=encrypt_window)
            if filepath:
                decrypt_file_wf(filepath, user['username'],password_key)
                messagebox.showinfo("Успіх", f"Файл {filepath} успішно розшифровано!",
parent=encrypt_window)



        def del_user(del_user_name, department):
                users = read_user_data()
                user_for_del = next(
                    (loc_user for loc_user in users if
                     loc_user['role'] == 'Regular user' and loc_user['department'] == department and
loc_user['username']==del_user_name), None)
                if user_for_del:
                    users.remove(user_for_del)
```

```
            write_user_data(users)
            os.remove(f"encrypted_keys/{department}/{user_for_del['username']}.key")
            try:
                os.remove(f"encrypted_biometrics/{department}/{user_for_del['username']}.npy")
            except:
                pass

    def add_user(department, username, nupassword):
        """Додає нового звич. користувача"""
        users = read_user_data()  # Зчитуємо існуючих користувачів

        # Перевіряємо, чи користувач з таким ім'ям вже існує
        if any(userl['username'] == username for userl in users):
            return False

        # Створюємо нового користувача
        new_user = {'username': username, 'password':
SHA512.new(nupassword.encode()).hexdigest(), 'role': "Regular user", 'department':department}
        users.append(new_user)  # Додаємо нового користувача до списку

        # Записуємо оновлений список користувачів у файл
        write_user_data(users)

        with open(f"{encrypted_key_folder}/{user['username']}.key",
                  "rb") as f:  # Читання зашифрованого ключа керівника
            encrypted_dep_key = f.read()
        saved_face_encoding = load_encrypted_biometric_data(user['username'],
                                                            password_key)  # Завантажуємо біометричні дані
керівника
        dep_key = decrypt_master_key_with_face(encrypted_dep_key, saved_face_encoding)
        # Створюємо ключ з пароля користувача
        nupassword_key = SHA256.new(nupassword.encode()).digest()

        # Шифрування головного ключа за допомогою пароля (користувача)
```

```
        cipher = AES.new(nupassword_key, AES.MODE_CBC)
        encrypted_new_man_key = cipher.encrypt(pad(dep_key, AES.block_size))

        # Зберігаємо зашифрований ключ
        with open(f"{encrypted_key_folder}/{username}.key", "wb") as f:
            f.write(cipher.iv + encrypted_new_man_key)
    def add_user_window():
        window = tk.Toplevel(encrypt_window)
        window.title("Додати користувача")
        window.iconbitmap("yeet.ico")
        window.geometry("300x320")

        tk.Label(window, text="Ім'я користувача:").pack(pady=5)
        username_entry = tk.Entry(window)
        username_entry.pack(pady=5)

        tk.Label(window, text="Пароль:").pack(pady=5)
        password_entry = tk.Entry(window, show="*")
        password_entry.pack(pady=5)

        def submit_user():
            username = username_entry.get()
            password = password_entry.get()
            department=user['department']
            add_user(department, username, password)
            window.destroy()
        def submit_user_del():
            del_user(username_entry.get(), user['department'])
            window.destroy()

        submit_user_button = tk.Button(window, text="Додати користувача",
command=submit_user)
        submit_user_button.pack(pady=20)
```

```
        del_button = tk.Button(window, text="Видалити користувача",
command=submit_user_del)
        del_button.pack(pady=20)
    if user['role'] != "Regular user":
        add_user_button = tk.Button(encrypt_window, text="Керування користувачами",
command=add_user_window, bg="#4CAF50", fg="white", font=("Helvetica", 12))
        add_user_button.pack(pady=20)
    # Кнопки для вибору файлів
    encrypt_button = tk.Button(encrypt_window, text="Зашифрувати файл",
command=encrypt_dialog, bg="#4CAF50",
                               fg="white", font=("Helvetica", 12))
    encrypt_button.pack(pady=20)

    decrypt_button = tk.Button(encrypt_window, text="Розшифрувати файл",
command=decrypt_dialog, bg="#2196F3",
                               fg="white", font=("Helvetica", 12))
    decrypt_button.pack(pady=20)
  else:
    saved_face_encoding = load_encrypted_biometric_data(user['username'],password_key)  #
Завантажуємо біометричні дані адміна

    def del_manager(department):
        users = read_user_data()
        manager = next(
            (loc_user for loc_user in users if
             loc_user['role'] == 'Manager' and loc_user['department'] == department), None)
        if manager:
            print("Заміна існуючого керівника.")
            users.remove(manager)
            write_user_data(users)
            os.remove(f"encrypted_keys/{department}/{manager['username']}.key")
            try:os.remove(f"encrypted_biometrics/{department}/{manager['username']}.npy")
            except:pass
```

```
    # додає керівника
    def add_manager(department, username, nmpassword):

        encrypted_key_folder = f"encrypted_keys/{department}"
        if not os.path.exists(encrypted_key_folder):
            os.makedirs(encrypted_key_folder)

        del_manager(department)
        users = read_user_data()
        if any(userl['username'] == username for userl in users):
            return False
        #Розшифрувати скопіювати та зашифрувати паролем керівника
        if os.path.exists(f"{encrypted_key_folder}/{user['username']}.key"):
            print("Головний ключ вже існує. Завантаження існуючого ключа")
            with open(f"{encrypted_key_folder}/{user['username']}.key", "rb") as f:# Читання
зашифрованого ключа адміна
                encrypted_dep_key = f.read()
            dep_key = decrypt_master_key_with_face(encrypted_dep_key, saved_face_encoding)   #
Розшифрування головного ключа адміна за допомогою обличчя


        else: #створення ключа деп., шифрув. обличчям адміна
            dep_key = get_random_bytes(32)    # Генерація випадкового головного AES ключа
            encrypted_dep_key = encrypt_master_key_with_face(dep_key, saved_face_encoding)
            with open(f"{encrypted_key_folder}/{user['username']}.key", "wb") as f:
                f.write(encrypted_dep_key)

        # Створюємо ключ з пароля керівника
        nmpassword_key = SHA256.new(nmpassword.encode()).digest()

        # Шифрування головного ключа за допомогою пароля (керівника)
        cipher = AES.new(nmpassword_key, AES.MODE_CBC)
        encrypted_new_man_key = cipher.encrypt(pad(dep_key, AES.block_size))
```

```
            # Зберігаємо зашифрований ключ керівника
            with open(f"{encrypted_key_folder}/{username}.key", "wb") as f:
                f.write(cipher.iv + encrypted_new_man_key)

            new_user = {'username': username, 'password':
SHA512.new(nmpassword.encode()).hexdigest(), 'role': 'Manager',
                        'department': department}
            users.append(new_user)
            write_user_data(users)
            print(f"Керівника департаменту {department} додано або замінено.")

        def add_manager_window():
            window = tk.Toplevel(encrypt_window)
            window.title("Додати керівника")
            window.iconbitmap("yeet.ico")
            window.geometry("300x320")

            tk.Label(window, text="Ім'я користувача:").pack(pady=5)
            username_entry = tk.Entry(window)
            username_entry.pack(pady=5)

            tk.Label(window, text="Пароль:").pack(pady=5)
            password_entry = tk.Entry(window, show="*")
            password_entry.pack(pady=5)

            tk.Label(window, text="Департамент:").pack(pady=5)
            department_entry = tk.Entry(window)
            department_entry.pack(pady=5)

            def submit():
                username = username_entry.get()
                password = password_entry.get()
                department = department_entry.get()
                add_manager(department, username, password)
```

```
            window.destroy()
        def submit_del():
            del_manager(department_entry.get())
            window.destroy()

        submit_button = tk.Button(window, text="Додати керівника", command=submit)
        submit_button.pack(pady=20)
        del_button = tk.Button(window, text="Видалити керівника", command=submit_del)
        del_button.pack(pady=20)

    def change_pass_window():
        window = tk.Toplevel(encrypt_window)
        window.title("Зміна пароля")
        window.iconbitmap("yeet.ico")
        window.geometry("300x320")

        tk.Label(window, text="Поточний пароль:").pack(pady=5)
        now_password_entry = tk.Entry(window, show="*")
        now_password_entry.pack(pady=5)

        tk.Label(window, text="Новий пароль:").pack(pady=5)
        new_password_entry = tk.Entry(window, show="*")
        new_password_entry.pack(pady=5)


        def submit():
            now_password = now_password_entry.get()
            new_password = new_password_entry.get()
            if user['password'] == SHA512.new(now_password.encode()).hexdigest():
                users = read_user_data()  # Читання даних користувачів
                users[0]["password"] = SHA512.new(new_password.encode()).hexdigest()
                save_encrypted_biometric_data(users[0]["username"],
SHA256.new(new_password.encode()).digest(), saved_face_encoding)
                root.destroy()
```

```
                with open(DATA_FILE, "w") as f:
                    json.dump(users, f, indent=4)
            else:
                messagebox.showerror("Помилка", "Невірний поточний пароль!", parent=window)

            window.destroy()

        submit_button = tk.Button(window, text="Підтвердити зміну пароля", command=submit)
        submit_button.pack(pady=20)

    change_pass_window = tk.Button(encrypt_window, text="Змінити пароль",
command=change_pass_window)
    change_pass_window.pack(pady=20)
    add_manager_button = tk.Button(encrypt_window, text="Додати чи видалити керівника",
command=add_manager_window)
    add_manager_button.pack(pady=20)
  """file window end"""



"""інтерфейс головного вікна"""
# Створення основного вікна
root = tk.Tk()
root.resizable(False, False)
root.title("Вхід")
root.iconbitmap("yeet.ico")
root.geometry("400x250")
root.configure(bg="#f0f0f0")

def create_entry_with_placeholder(parent, placeholder):
  entry = tk.Entry(parent, font=("Helvetica", 12), width=25)
  entry.insert(0, placeholder)
  entry.bind("<FocusIn>", lambda e: clear_placeholder(entry, placeholder))
```

```
    entry.bind("<FocusOut>", lambda e: restore_placeholder(entry, placeholder))
    return entry

def clear_placeholder(entry, placeholder):
    if entry.get() == placeholder:
        entry.delete(0, 'end')

def restore_placeholder(entry, placeholder):
    if entry.get() == '':
        entry.insert(0, placeholder)



frame = tk.Frame(root, bg="#f0f0f0", borderwidth=1, relief="raised")
frame.place(relx=0.5, rely=0.5, anchor='center')
# Поля введення
username_entry = create_entry_with_placeholder(frame, "Ваш логін")
username_entry.pack(pady=5)

password_entry = create_entry_with_placeholder(frame, "Ваш пароль")
password_entry.config(show="*")  # Показувати замість пароля
password_entry.pack(pady=5)



button_frame = tk.Frame(frame, bg="#f0f0f0")  # окрема рамка для кнопок
button_frame.pack(pady=15)
# Кнопки
login_button = tk.Button(button_frame, text="Увійти", command=login, bg="#4CAF50",
fg="white",
                         font=("Helvetica", 12), width=15)
login_button.pack(side=tk.LEFT, padx=5)

"""інтерфейс головного вікна end"""
# Запуск головного циклу
root.mainloop()
```