Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом”

Виконав здобувач вищої освіти
IV курсу, групи КБ-22-МБ
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Кваша М.В.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук
_____ Смірнова Т.В.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Кваші Михайлу Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом*

2. Керівник роботи *Смірнова Тетяна Віталіївна, канд. техн. наук*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 51-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи кібербезпеки в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи кібербезпеки	*1 аркуш*
Функціональна схема системи кібербезпеки	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Смірнова Т.В.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Кваша М.В.
(прізвище та ініціали)

АНОТАЦІЯ

Кваша М.В. Програмне забезпечення системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

Метою розробки є програмне забезпечення системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

Результат роботи – програмна реалізація системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: кібербезпека, електронне посвідчення, система контролю й управління доступом

ABSTRACT

Kvasha M.V. Software for the cybersecurity system of implementing an electronic certificate for access control and management systems. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the cybersecurity system of implementing an electronic certificate for access control and management systems.

The purpose of the development is the software for the cybersecurity system of implementing an electronic certificate for access control and management systems.

The result of the work is the software implementation of the cybersecurity system of implementing an electronic certificate for access control and management systems.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software are provided.

The program can be used on PCs with Windows 10/11.

The program is developed in the Python environment.

Keywords: cybersecurity, electronic identity, access control and management system

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АСУ – автоматизована система управління

ДСТ – держстандарт

ЕМЗ – електромагнітний замок

ЕОМ – електронно-обчислювальна машина

ІСО – міжнародний стандарт

ОПС – охоронно-пожежна сигналізація

ПЗ – програмне забезпечення

ПЗП – постійний запам’ятовуючий пристрій

ПК – програмний комплекс

СКУД – система контролю та управління доступом

СУД – система управління доступом

PIN – personal identification cod – персональний ідентифікаційний код

RTE – Request To Exit – кнопка «Вихід»

ВСТУП

Актуальність теми. Електронне посвідчення поєднує функції декількох карт і пропусків. Це рішення на базі універсальної смарт-карти сполучає пропуск на територію організації або в приміщення, засіб доступу в інформаційні системи компанії, персональний засіб електронного підпису (ЕП), а також платіжну або зарплатну карту. На карті також можуть бути фотографія, ПІБ, дата народження й ID співробітника, що робить її ще більш зручною у використанні.

Електронне посвідчення інтегрується із системами управління фізичним доступом. Завдяки можливості розміщення відразу двох RFID-міток, ця карта може використовуватися як єдиний пропуск на два об'єкти організації (наприклад, у центральний і регіональний офіси), а її інтеграція із системою контролю й управління доступом підприємства дає можливість здійснювати моніторинг переміщень співробітників по приналежні йому території, а також вести облік робочого часу. Електронне посвідчення може служити як персональний засіб ЕП з ключем, що не витягається, ЕП і сертифікованим програмним інтерфейсом для візуалізації документа, що підписується, що дозволяє забезпечити юридичну значимість електронного документообігу компанії у відповідності з усіма вимогами. Підвищення рівня безпеки корпоративних IT-ресурсів досягається за рахунок використання дво- або трифакторної автентифікації при доступі до інформаційних систем. Рішення дозволяє значно скоротити строки й витрати на впровадження, експлуатацію й підтримку систем безпеки за рахунок використання єдиної системи управління засобами автентифікації й ЕП на підприємстві.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем реалізації електронного посвідчення для систем контролю й управління доступом.

– Дослідження системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

– Програмна реалізація системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі реалізації електронного посвідчення для систем контролю й управління доступом.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система контролю й управління доступом (**СКУД**) є одним з компонентів системи безпеки. Вона може застосовуватися як самостійно, так і в комплексі (інтегровано) з іншими компонентами системи безпеки, наприклад, з охоронної й пожежної сигналізаціями, системою охоронного телебачення. У кожному разі основним завданням СКУД є організація переміщення через контрольні точки (точки доступу) за певними правилами. Як суб'єкт можуть виступати як фізичні особи, так і транспортні засоби

Насправді в СКУД відслідковується якийсь ідентифікатор, прив'язаний до суб'єкта й визначального його права. Як ідентифікатор можуть застосовуватися як фізичні носії, наприклад, ключ-“таблетка”, безконтактна карта, брелок, так і коди, що вводяться із клавіатури. Крім цього для ідентифікації людини використовуються його біометричні ознаки, наприклад відбиток пальця, рисунок сітківки або райдужної оболонки ока. А для ідентифікації автомобіля може бути використано, наприклад, його відео-зображення. Іноді, для підвищення безпеки використовують сукупність відразу декількох ідентифікаторів. Як точки доступу можуть виступати пристрою, що обмежують прохід (проїзд) об'єкта, такі як двері, хвіртка, турнікет, шлагбаум, ворота. Застосовуються й більше складні точки контролю доступу, у яких пристрої, що перепиняють, об'єднані загальною логікою роботи, – це шлюзи. Роботу СКУД у загальному випадку можна представити як ідентифікацію суб'єкта й надання або заборона йому прав на прохід (проїзд) через контрольовану точку. Рішення про це приймає контролер, що є “мозком” системи

1.2 Область застосування

Автономні СКУД

Найпростішим випадком СКУД є автономна система. У цьому випадку контролер управляє проходом через закріплені за ним точки доступу (як правило, одна або дві). Саме він зберігає у своїй пам'яті інформацію про ідентифікатори й приналежних їм правах. Автономна система в загальному випадку складається із пристрою, що зчитує ідентифікаційну ознаку, контролера й виконавчого механізму (замок, турнікет і т.п.). Такі системи відносно недорогі й у теж час надійні. Тому вони одержали широке поширення на об'єктах з невеликою кількістю точок доступу, де не потрібне відстеження переміщень суб'єкта по об'єкті

Мережна Система Контролю й Управління Доступом

Однак якщо кількість точок доступу значно й вони повинні бути об'єднані загальною логікою роботи, доводиться застосовувати мережні СКУД. У мережній системі всі контролери об'єднані один з одним через комп'ютер, і інформація про події від всіх точок доступу, як правило, обробляється в одному місці. Це дає можливість використовувати додаткові можливості в порівнянні з автономними системами. Наприклад, можна заборонити прохід через певну точку доступу поки не пройдена попередня. Тобто мережні СКУД дозволяють не тільки відслідковувати переміщення суб'єкта, але й визначати правила цих переміщень. Вся інформація зберігатися в БД і це дає можливість контролювати події, які відбувалися в минулому (перегляд архіву), а також оперативно втручатися в роботу системи в реальному часі. Наприклад, оператор може примусово заборонити (дозволити) прохід через конкретну точку доступу. До важливої переваги мережні СКУД варто віднести можливість організації автоматизованого обліку робочого часу й контролю трудової дисципліни. І нарешті, безумовним достоїнством мережних СКУД є можливість їхньої інтеграції з іншими компонентами системи безпеки: охоронно-пожежною сигналізацією, системою охоронного телебачення.

Інтеграція СКУД з іншими компонентами системи безпеки

Спільне використання різних компонентів системи безпеки дозволяє істотно підвищити її ефективність.

Так, наприклад, інтеграція СКУД з охоронним відеоспостереженням дає можливість відеоверифікації суб'єкта в момент проходу через точку доступу. Тобто можна зрівняти зображення суб'єкта з камери в момент проходу із зображенням власника ідентифікатора, занесеного в базу даних. Крім того, така інтеграція дозволяє більш раціонально організувати відеозапис у зоні точки доступу, прив'язавши її до події проходу. Це також набагато спрощує пошук конкретних подій по відеоархіві

Інтеграція СКУД і охоронно-пожежної сигналізації (**ОПС**)підвищує як захищеність, так і безпека об'єкта. Так, наприклад, можна блокувати доступ у приміщення, що перебувають під охороною. Крім того, можна організувати постановку/зняття системи під охорону, використовуючи ідентифікатор, зареєстрований у СКУД. У випадку спрацювання пожежної сигналізації автоматичне розблокування перешкод на шляхах евакуації значно підвищує шанс на порятунок.

Технічне обслуговування СКУД

Сучасні системи контролю й управління доступом являють собою досить складну інженерну систему, що складається з багатьох компонентів. Тут сполучаються як апаратні, так і програмні засоби. Тому для підтримки системи в робочому стані потрібен кваліфікований персонал, що робить планово-профілактичні й ремонтно-відбудовчі роботи.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Готовий комплект СКУД: Офіс – 1 двері

Готовий комплект «Офіс», на базі СКУД «Castle» являє собою набір апаратно-технічні засоби для контролю управління доступом на 1 двері, з функціями контролю робочого часу.

Готовий комплект поставляється в одній коробці, має закінчений товарний вид, утримуються детальні інструкції з підключення більше 100 пристроїв серед яких: зчитувачі, турнікети, електрозамки. Комплект містить у собі все необхідне для підключення системи контролю й управління доступом в офісі, розрахованому наприклад на 50 чоловік.

Система контролю й управління доступом (СКУД) – це сучасний помічник служби безпеки й охорони, що складається з різних контролерів, турнікетів, електронних і кодових замків, зчитувачів карт доступу й шлагбаумів. При установці контролю доступу кожний співробітник одержує персональний ідентифікатор – картку певного формату, на якій можуть бути нанесені додаткові ідентифікатори, або одержує доступ на підставі індивідуального відбитка пальця (біометрична СКУД). За допомогою різних видів зчитувачів (антивандальні, далекобійні, мініатюрні), співробітник одержує доступ тільки до певних заданих об'єктів (точкам проходу).

Сучасна СКУД «Castle» є мережною, багатокористувальницькою системою контролю доступу. Система «Castle» розроблена з обліком кращого світового досвіду. Основою акцент при розробці СКУД «Castle» робився на надійність системи.

Адміністрування й збір інформації з контролерів здійснюється за допомогою програмного забезпечення «Castle». Контролери «Castle» можуть необмежено довго продовжувати функціонувати при відсутності зв'язку, без запуску програмного забезпечення. Кожний контролер при цьому автономно запам'ятовує до 40 000 подій проходу в енергонезалежній пам'яті. При запуску програмного забезпечення події автоматично надходять у базу даних на комп'ютер. Гарантія на встаткування під маркою «Castle» становить 5 років.

Контроль доступу в приміщення офісу для декількох дверей і турнікетів

Система комплексного захисту великого офісу базується на широкому використанні сучасних цифрових технологій в області автоматизації й системної інтеграції. Залежно від складності такої системи й необхідних функціональних можливостей монтаж систем контролю доступу припускає оснащення приміщення або території камерами відеоспостереження, електронними замками, для відкриття яких можуть використовуватися спеціальні картки доступу, а також оснащення пропускних пунктів автоматичними турнікетами.

Для цього приклада нами був обраний середній офісний будинок, хоча це може бути й середній готель на 70-80 номерів, вхід у яке здійснюється через прохідну, на якій ми встановили 2 турнікети. Всі двері в будинку під надійною охороною й відкриваються електронно за допомогою карток.

Обов'язковим елементом системи контролю доступу на великих підприємствах і режимних об'єктах, є турнікет, що фізично обмежує доступ на підприємство осіб, що не мають на це відповідних дозволів. Конструкції й електронна начинка сучасних турнікетів дозволяють наділяти їх самими широкими функціональними можливостями, більше того СКУД «Castle» має можливість працювати з будь-якими типами турнікетів і задавати будь-яку логіку їхньої роботи.

Контроль доступу у великому офісі може бути не тільки засобом забезпечення безпеки, але й ефективним інструментом управління

підприємством, наприклад готелем. У цьому випадку обов'язковим елементом системи стає сервер безпеки, за допомогою якого здійснюється контроль за ефективністю використання робочого часу персоналу. За допомогою сервера безпеки також можна створити багатоступінчасту багаторівневу систему моніторингу діяльності підприємства. Зі СКУД «Castle» зробити такий сервер безпеки зручно й дуже вигідно, тому що клієнтські місця не вимагають окремого ліцензування.

Організація КПП, організація прохідний турнікетами

Для організації контролю доступу на підприємство з використанням безконтактних карт допуску можна використовувати готовий комплект на базі СКУД «Castle» – «Великий офіс 2», пропонується встановити 4 турнікети зі зчитувачами безконтактних карт доступу, 2 службові двері захистити за допомогою магнітних замків і також наділити їх зчитувачами.

Автоматизована прохідна забезпечує контроль проходу у двох напрямках у місцях з великим потоком співробітників, із цим завданням легко справляється СКУД «Castle». Турнікет ідеальний для контролю доступу в будь-яких випадках, коли необхідно сполучити в одне ціле максимальну надійність, комфорт і різний дизайн (завдяки розширеній інтеграції турнікетів у СКУД «Castle»).

Турнікет ОМА-26.461 легко підключається до СКУД «Castle»

Турнікет призначений для управління потоками людей при посиленому контролі доступу. Надійно перекриває прохід і розділяє потік людей по одному. Дуже компактний, але при цьому забезпечує коректний поділ потоку людей «по одному», при регулюванні доступу на територію. Подовжений корпус (505 мм) дозволяє обійтися без додаткових огороджень зони проходу. Корпус турнікета можна вішати на стійки огородження або на стіну. На бічні поверхні турнікета легко встановити будь-який зчитувач. Турнікет трипод ОМА-26.461 повністю управляється дистанційно й ідеально підходить для організації прохідних підприємств із посиленим двонаправленним контролем доступу на територію об'єкта (на вхід і вихід), як при автономному режимі роботи (ручне управління з

пульта охоронцем з візуальним контролем ситуації), так і при роботі турнікета як виконавчий пристрій у системі контролю й управління доступом (СКУД) – автоматизована прохідна, коли необхідна чітка автоматична реєстрація числа й напрямку проходів.

СКУД з підключенням турнікета, воріт, і двох дверей

Є презентабельний особняк із прилягаючою територією. Територія контролюється системою відеоспостереження. На вході є двері й ворота для в'їзду автомашин на паркування. У самому будинку встановлений турнікет, а також узятий під охорону службовий вхід. При в'їзді/виїзді можуть установлюватися далекобійні зчитувачі, які передають код кодоносія, установленого в машині в комп'ютер. Ця інформація дозволяє визначити номер, марку, графік руху, вантажі, для яких використовується й т.д. Також у системі може бути передбачений контроль часу доставки вантажу до воріт (якщо інформація про час виїзду буде вводитися в систему документообігу). Це робиться для контролю ситуації: «одне вивантажили – інше завантажили». Наявність зв'язку між системою контролю доступу й системою документообігу дозволяє виключити можливість появи липових накладних.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування

Python – динамічна інтерпретована об’єктно-орієнтована скриптова мова програмування із строгою динамічною типізацією. Офіційний сайт мови програмування Python https://www.python.org/. Python – багатоцільова мова програмування, яка дозволяє писати код, що добре читається. Відносний лаконізм мови Python дозволяє створити програму, яка буде набагато коротше свого аналога, написаного на іншій мові. Python – багатоплатформова мова програмування. Це означає, що програми на Python можна запускати в різних операційних системах без будь-яких змін.

Ще однією перевагою Python є його стандартна бібліотека, яка встановлюється разом з Python і містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних, для побудови графічного інтерфейсу програм тощо. Програми, написані на мові програмування Python, можуть бути як невеликими скриптами, так і складними системами. Python абсолютно безкоштовний.

Швидкість виконання коду Python

Один з можливих недоліків Python – швидкість виконання коду. Python не є компільованою мовою. Код на Python спочатку компілюється у внутрішній байт-код, який потім виконується інтерпретатором Python. У більшості випадків при використанні Python виходять програми повільніші в порівнянні з такими мовами, як C.

Втім, сучасні комп'ютери мають таку обчислювальну потужність, що для більшості застосунків швидкість розробки важливіша швидкості виконання, а програми на Python зазвичай пишуться набагато швидше.

Окрім того, Python легко розширюється модулями, написаними на C або C++. Такі модулі можуть використовуватися для виконання частин програми, що створюють інтенсивне навантаження на процесор.

Використання Python

Python використовується для різних цілей: для створення ігор і веб-застосунків, розробки внутрішніх інструментів для різноманітніх проектів. Мова також широко застосовується в науковій області для досліджень і розв'язування прикладних завдань.

Застосування мови програмування Python:

1. BitTorrent – протокол для обміну даними.

2. Ubuntu Software Center – вільне програмне забезпечення для пошуку, установки і видалення пакунків в системі Ubuntu Linux.

3. Blender – програма для створення тривимірної комп'ютерної графіки, що включає засоби моделювання, анімації, вимальовування, пост-обробки відео, а також створення відеоігор.

4. GIMP – растровий графічний редактор, із підтримкою векторної графіки.

5. World of Tanks.

6. Вільна енциклопедія Вікіпедія.

7. Пошукова система Google.

8. DropBox – файловий хостинг, що включає персональне хмарне сховище, синхронізацію файлів і програму-клієнт.

9. YouTube – популярне відеосховище.

Версії Python

Мови програмування з часом змінюються – розробники додають в них нові можливості, а також виправляють помилки. Так з'являються різні версії мови. Наприклад, код написаний на Python 2 у більшості випадків не буде працювати у версії Python 3 без внесення додаткових змін.

Процесор є найважливішим компонентом в комп'ютері. Одна з основних функцій процесора – це обробка даних згідно комп'ютерної програми, яка є списком інструкцій, шляхом виконання арифметичних і логічних операцій над фрагментами даних.

Кожна інструкція в програмі – це команда, яка «повідомляє» процесору, яку операцію він повинен виконати. Процесор комп'ютера може розуміти лише ті інструкції, які написані на машинній мові. Машинна мова – це штучна мова, створена для передачі команд комп'ютеру. За допомогою машинної мови створюються ефективні програми, оскільки розробник отримує доступ до всіх можливостей процесора. Машинна мова – мова низького рівня.

Інструкція машинної мови існує для кожної операції, яку процесор здатний виконати – є інструкція для додавання чисел, є інструкція для віднімання

чисел і т.д. Увесь набір інструкцій, який центральний процесор може виконати, відомий як набір інструкцій процесора.

Наприклад, у вас є певна програма, яка зберігається на диску вашого комп'ютера. Для виконання програми, ви здійснюєте подвійний клік на значку програми. Це змушує програму копіюватися з диска в оперативну пам'ять, після чого процесор комп'ютера виконує копію програми, яка знаходиться в оперативній пам'яті.

Коли процесор виконує інструкції програми, він бере участь у процесі, який є відомим як цикл fetch – decode – execute (отримати – декодувати – виконати). Цей цикл виконується для кожної інструкції у програмі і складається з трьох кроків:

Отримати

Програма – це послідовність інструкцій на машинній мові. Першим кроком циклу є завантаження (отримання) наступної інструкції з пам'яті в процесор.

Декодувати

Інструкція машинної мови – це двійкове число, яке представляє команду, що повідомляє процесору виконати певну операцію. На цьому кроці процесор декодує інструкцію, яку було «витягнуто» з пам'яті, для визначення того, яка операція повинна виконуватись.

Виконати

Останній крок циклу – виконати операцію.

Хоча процесор комп'ютера розуміє тільки машинну мову, людині непрактично писати програми на машинній мові. Така програма може мати тисячі або навіть мільйони бінарних інструкцій, і написання такої програми буде дуже обтяжливим процесом.

З цієї причини була створена мова асемблера як альтернатива машинній мові. Замість використання двійкових чисел для написання інструкцій, мова асемблера використовує короткі слова, відомі як мнемокоди.

Незважаючи на те, що мова асемблера не вимагає двійкових інструкцій, як у випадку машинної мови, проте вона вимагає високих знань про процесор. Використовуючи мову асемблера, навіть для найпростішої програми, необхідно написати велику кількість інструкцій.

Мова програмування високого рівня дозволяє створювати складні програми, не знаючи, як працює процесор, і не записуючи великої кількості інструкцій низького рівня. Крім того, більшість мов програмування високого рівня використовують слова, які легко зрозуміти.

Python – одна із популярних сучасних мов програмування високого рівня. Python – інтерпретована мова програмування. Python – це високорівнева інтерпретована мова програмування, на відміну від C++, яка є прикладом компільованої мови програмування. Назва Python відноситься як до мови програмування, так і до інтерпретатора – комп'ютерної програми, яка зчитує початковий код (написаний на Python) і виконує інструкції (команди).

Для перекладу мови високого рівня на машинну мову доступні два типи програм:

1. Компілятор.

2. Інтерпретатор.

Завантаження Python

Версії інтерпретатора Python для різних операційних систем доступні для безкоштовного завантаження за адресою https://www.python.org/downloads.

Середовище програмування для Python

Для написання програм використовують текстові редактори або інтегровані середовища розробки, які включають в себе різні інструменти для роботи з кодом: засіб для написання коду (текстовий редактор), інтерактивний інтерпретатор, відлагоджувач тощо.

Текстові редактори та інтегровані середовища програмування для Python:

– IDLE – стандартний редактор Python. Встановлюється разом з Python для користувачів Windows, окремим пакунком для користувачів Linux.

– Notepad++ – безкоштовний текстовий редактор початкового коду, який підтримує велику кількість мов, в тому числі і Python. Лише для користувачів Windows.

– Visual Studio Code – це легкий, але потужний редактор початкового коду, який розповсюджується безкоштовно і доступний у версіях для платформ Linux, Windows і macOS.

– PyScripter – інтегроване середовище розробки для мови програмування Python. Для користувачів Windows. Поширюється безкоштовно.

– Wing IDE 101 – вільне інтегроване середовище для Python, розроблене для навчання програмістів-початківців. Для користувачів Linux, Windows і macOS. Поширюється безкоштовно.

– Geany – вільний текстовий редактор з базовими елементами інтегрованого середовища розробки, доступний для операційних систем Linux, Windows і macOS.

– PyCharm – інтегроване середовище розробки для мови програмування Python. PyCharm є власницьким програмним забезпеченням. Наявна безкоштовна версія Community з усіченим набором можливостей. Для користувачів Linux, Windows і macOS.

– Thonny – IDE для вивчення програмування мовою Python. Для користувачів Linux, Windows і macOS.

– Mu – редактор коду Python для програмістів-початківців. Для користувачів Linux, Windows і macOS.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи кібербезпеки контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи кібербезпеки в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Контроль доступу – це фундаментальна концепція безпеки, яка використовується для регулювання того, хто може переглядати або використовувати ресурси в обчислювальному середовищі. Це гарантує, що лише авторизовані особи або системи мають доступ до даних або систем, таким чином захищаючи конфіденційну інформацію та зберігаючи конфіденційність, цілісність і доступність критично важливих ресурсів в організації.

За своєю суттю контроль доступу передбачає ідентифікацію, автентифікацію та авторизацію користувачів, гарантуючи, що дозволи доступу відповідають політикам організації та ролям користувачів. У своїй найпростішій формі контроль доступу можна розглядати як комбінацію паролів і дозволів користувача; однак сучасні системи контролю доступу включають передові технології, такі як біометрична перевірка, багатофакторна автентифікація та динамічні рамки на основі політики. Ці системи мають важливе значення для керування безпекою в різноманітних середовищах, починаючи від великих підприємств і закінчуючи малими підприємствами, і мають вирішальне значення для дотримання нормативних стандартів, одночасно захищаючи від порушень даних і спроб несанкціонованого доступу.

Що таке системи контролю доступу?

Контроль доступу – це тип системи безпеки, яка обмежує доступ до просторів або систем користувачам, які отримали дозвіл і інструменти для отримання доступу, як-от ключі або картки-ключі для фізичних просторів і облікові дані для цифрових активів. Ця робота буде зосереджена на контролі доступу користувачів, який забезпечує доступ до обмежених зон підприємства (наприклад, систем і програм).

Контроль доступу користувачів працює на основі принципів перевірки особи та керування дозволами для забезпечення виконання політик безпеки. Коли особа або система запитує доступ до ресурсу, елементи керування доступом користувача перевіряють особу запитувача та автентифікують його.

За допомогою цих цифрових засобів контролю доступу користувач спочатку має бути ідентифікований та автентифікований, перш ніж отримати доступ до приватної інформації, що означає, що основи системи контролю доступу включають критерії та записи кожного разу, коли хтось «входить» у систему. Залежно від типу організації, підприємству слід розглянути кілька загальних ідей – який рівень власності воно матиме на систему та як вирішити, хто з працівників отримає до чого доступ. Існує багато моделей, кожна з яких має різні переваги.

Які є три основні типи систем контролю доступу?

Існує кілька типів систем контролю доступу, але три основних:

1. Системи обов'язкового контролю доступу (MAC).
2. Дискреційні системи контролю доступу (DAC).
3. Рольові системи контролю доступу (RBAC).

Обов'язковий контроль доступу (MAC)

Система обов’язкового контролю доступу забезпечує найсуворіший захист безпеки, де право дозволяти доступ повністю належить системним адміністраторам. Це означає, що користувачі не можуть змінювати дозволи, які забороняють або дозволяють їм вхід у різні зони, створюючи потужний захист конфіденційної інформації. Це навіть обмежує можливість власника ресурсу надавати доступ до всього, що вказано в системі.

Коли працівник входить до системи, він позначається унікальним з’єднанням змінних «тегів», як-от профіль цифрової безпеки, що вказує на те, який рівень доступу він має. Отже, залежно від того, які теги має користувач, він матиме обмежений доступ до ресурсів на основі конфіденційності інформації, що міститься в ньому. Ця система безпеки насправді настільки хитра, що її зазвичай

використовують державні установи через її зобов'язання щодо конфіденційності.

Дискреційний контроль доступу (DAC)

Дискреційна система контролю доступу, з іншого боку, повертає трохи більше контролю в руки керівництва безпеки. Вони визначають, хто може отримати доступ до яких ресурсів, навіть якщо системний адміністратор створив ієрархію файлів з певними дозволами. Щоб отримати доступ, потрібні лише правильні облікові дані.

Єдиний недолік, звичайно, полягає в тому, що для кінцевого користувача потрібен контроль над рівнями безпеки. А оскільки система вимагає більш активної ролі в управлінні дозволами, легко дозволити діям провалитися. Там, де підхід MAC є жорстким і не потребує зусиль, система ЦАП є гнучкою та вимагає великих зусиль.

Контроль доступу на основі ролей (RBAC)

Контроль доступу на основі ролей надає користувачеві дозволи на основі його бізнес-обов'язків. Як найпоширеніша система контролю доступу, вона визначає доступ на основі ролі користувача в компанії, допомагаючи гарантувати, що працівники нижчого рівня не отримають доступу до інформації високого рівня.

Права доступу в цьому методі розроблені навколо набору змінних, які відображаються назад до бізнесу, наприклад ресурсів, потреб, середовища, роботи, місця розташування тощо. Багатьом керівникам подобається цей підхід, оскільки легко групувати співробітників на основі ресурсів, до яких їм потрібен доступ.

Наприклад, комусь із відділу кадрів не потрібен доступ до приватних маркетингових матеріалів, а працівникам маркетингу не потрібен доступ до зарплат співробітників. RBAC забезпечує гнучку модель, яка покращує видимість, зберігаючи захист від порушень безпеки та витоку даних.

Які чотири основні моделі контролю доступу?

Існує чотири основні моделі контролю доступу, які поділяються на два

загальні типи. Детальніші, практичні моделі контролю доступу пропонують можливість для більш індивідуальних підходів. Залежно від того, наскільки «практичним» підприємство хоче бути, є багато способів думати про це. Два найпоширеніші типи засобів контролю доступу – це керування доступом на основі правил і керування доступом на основі атрибутів.

«Розумніші», більш інтуїтивно зрозумілі системи контролю доступу виходять за межі технологій. Це системи, які працюють на більш глибокому, більш інтуїтивному рівні. Прикладами такого типу керування доступом є керування доступом на основі ідентичності та керування доступом на основі історії.

Контроль доступу на основі правил

Як ви могли здогадатися, ця система надає дозволи на основі структурованих правил і політик. Здебільшого на основі контексту, коли користувач намагається отримати доступ до ресурсу, операційна система перевіряє правила, вибрані в «списку контролю доступу» для цього конкретного ресурсу. Створення правил, політик і контексту додає певних зусиль до розгортання. Крім того, ця система часто поєднується з підходом на основі ролей, який ми обговорювали раніше.

Контроль доступу на основі атрибутів

Переходячи на глибший рівень, цей тип системи забезпечує різний динамічний і розумний контроль на основі атрибутів, наданих конкретному користувачеві. Думайте про ці атрибути як про компоненти профілю користувача; разом вони визначають доступ користувача.

Після встановлення політики вони можуть використовувати ці атрибути, щоб дізнатися, чи повинен користувач мати контроль. Ці атрибути також можна отримати та імпортувати з окремої бази даних, як-от Salesforce, наприклад.

Контроль доступу на основі ідентифікації

Найпростіший, але найскладніший – контроль на основі ідентифікації визначає, чи дозволено користувачеві доступ до ресурсу на основі його

індивідуальної візуальної чи біометричної ідентифікації. Тоді користувачеві буде заборонено або дозволено доступ залежно від того, чи можна зіставити його особу з іменем у списку контролю доступу.

Однією з головних переваг цього підходу є надання більш детального доступу до окремих осіб у системі на відміну від групування співробітників вручну. Це дуже детальний технологічний підхід, який дає власнику бізнесу велику кількість контролю.

Контроль доступу на основі історії

Ще одне «розумне» рішення – система контролю доступу на основі історії. На основі попередніх дій безпеки система визначає, чи отримує користувач доступ до ресурсу, який він запитує.

Потім система збирає історію дій цього користувача – час між запитами, запитуваний вміст, які двері нещодавно відкривалися тощо. Наприклад, якщо користувач довгий час працював виключно із захищеними бухгалтерськими матеріалами, запит на доступ до маркетингової дорожньої карти наступного року може бути позначено в системі.

Ключові компоненти систем контролю доступу

Системи контролю доступу базуються на трьох основних компонентах – ідентифікації, автентифікації та авторизації. Кожен крок має бути успішно виконано, перш ніж користувачі отримають доступ до ресурсів.

Адміністратори видають унікальну ідентифікацію кожному користувачеві, коли він підключається до систем або програм. У разі обговорюваних тут елементів керування доступом користувачів вони використовуються для ідентифікації, а також для відстеження дій користувачів.

Автентифікація проти авторизації

Перш ніж користувачі зможуть отримати доступ до систем або програм, вони повинні пройти автентифікацію. Автентифікація – це процес безпеки для підтвердження того, що особа, представлена користувачем, є законною. Основні типи факторів автентифікації:

– Щось, що ви знаєте. Найпоширенішим прикладом цього є облікові дані користувача та пароля.

– Щось у вас є. Часто це мобільний пристрій, на який надсилаються коди підтвердження. Це також може бути апаратний маркер або програма, яка генерує одноразові паролі.

– Щось, що ви є – це біометрія, наприклад відбитки пальців, сканування райдужної оболонки ока або розпізнавання обличчя.

Після перевірки користувачів їм надається доступ до ресурсів на основі авторизації. Авторизація визначає, що може робити автентифікований користувач після того, як йому надано доступ до системи чи програми.

Наприклад, користувач із широкими правами може отримати повний доступ до ресурсів і мати можливість не лише створювати, а й редагувати, ділитися або видаляти існуючі файли та інші ресурси. Адміністратори також налаштовують це кількома способами, як пояснено в розділах вище (наприклад, контроль доступу на основі ролей і контроль доступу на основі атрибутів).

Кращі практики впровадження систем контролю доступу

Застосовуйте принцип найменших привілеїв (PoLP)

Весь доступ має надаватися на основі найменших привілеїв. Це означає надання користувачам мінімальних привілеїв доступу, необхідних для виконання їхньої роботи. Він також вимагає відкликати привілеї, коли доступ більше не потрібен.

Автоматизуйте надання доступу користувачам

Автоматизація ініціалізації та скасування ініціалізації користувачів покращує контроль доступу, усуваючи ризики, пов’язані з ручними процесами, як-от затримки між моментом, коли користувач залишає організацію, та моментом, коли його привілеї доступу відкликаються. Це також допомагає забезпечити послідовне виконання політик доступу.

Проводьте регулярні перевірки доступу

Необхідно запровадити процеси для забезпечення регулярного проведення

аудитів систем контролю доступу. Це допомагає забезпечити дотримання принципу найменших привілеїв шляхом ідентифікації користувачів із надлишковими привілеями та ідентифікації облікових записів користувачів, які неактивні.

Забезпечте інтеграцію з існуючими програмами

Виберіть систему контролю доступу, яка підтримує інтеграцію з додатками та системами, що використовуються в організації, включаючи інші системи безпеки. Це не тільки полегшує керування, але й дозволяє уникнути громіздких процесів, які заважають продуктивності користувачів або, що ще гірше, змушують їх шукати обхідні шляхи, щоб обійти стомлюючі системи контролю доступу.

Реалізація мережевої архітектури нульової довіри (ZTNA)

Використовуйте архітектуру нульової довіри для керування пристроями, мережами, програмами, службами, робочими навантаженнями та даними. Потім запровадьте основні принципи нульової довіри, зокрема:

– Постійно відстежуйте та перевіряйте користувачів, постійно перевіряючи особистість і привілеї користувачів після надання початкового доступу.

– Встановіть політики доступу відповідно до принципу найменших привілеїв, регулярно оцінюючи та оновлюючи привілеї в міру зміни вимог користувачів.

– Використовуйте мікросегментацію для поділу мереж, зменшення поверхні атаки шляхом обмеження доступу до невеликих областей і ресурсів.

– Вимагайте багатофакторної автентифікації для покращення контролю доступу користувачів.

– Перевірте всі кінцеві пристрої, розширивши контроль доступу до фізичних систем на додаток до користувачів.

Заборонити використання спільних облікових записів

Настійно рекомендовано, щоб користувачам ніколи не дозволялося

ділитися обліковими записами для систем доступу або програм. Це створює ризики безпеці, оскільки ускладнює відстеження активності облікового запису та притягнення користувачів до відповідальності за їхні дії або відстеження першопричин у разі інциденту безпеки, пов’язаного зі спільним обліковим записом.

Використовуйте розподіл обов'язків

Це стосується розподілу обов’язків, завдань і відповідальності між кількома користувачами, щоб гарантувати, що жодна особа не має достатніх привілеїв для неправильного використання ресурсів. Класичним прикладом цього є заборона особі, яка виписує чеки, також їх підписувати. Розподіл обов’язків застосовується до широкої бази користувачів, за винятком привілейованих користувачів, таких як адміністратори.

Вимагати надійних паролів

Від користувачів слід вимагати використання надійних паролів. Рекомендації Агентства з кібербезпеки та безпеки інфраструктури (CISA) щодо надійних паролів:

– Зробіть їх довгими – принаймні 16 символів.

– Зробіть їх довільними за допомогою рядка змішаних регістрових літер, цифр і символів, наприклад cXmnZK65rf*&DaaD, або використовуйте рядок непов’язаних слів, щоб створити парафраз, наприклад LibraryBeachBananaFlower.

– Зробіть їх унікальними.

– Використовуйте різні паролі для кожного облікового запису.

Використовуйте багатофакторну автентифікацію (MFA)

Вимагайте від користувачів надати більше ніж один фактор перевірки, щоб отримати доступ, як-от облікові дані користувача та біометричний фактор або одноразовий пароль, створений із маркера або програми для створення коду автентифікації.

Ведіть журнали привілеїв доступу

Використовуйте журнали привілеїв доступу, щоб відстежувати, які

користувачі мають доступ до яких ресурсів, і відстежувати зміни в доступі користувачів.

Навчіть співробітників протоколам контролю доступу

Додайте засоби контролю доступу до програм навчання та підвищення обізнаності з кібербезпеки, щоб гарантувати, що користувачі розуміють, які системи існують, чому вони використовуються та ризики недотримання правил використання.

Майбутнє: управління ідентифікацією за допомогою ШІ

Оскільки контроль доступу рухається в майбутнє, багато хто прогнозує, що відповідальність за управління системами й надалі переходитиме від людей до технологій. Штучний інтелект (AI) не тільки дозволяє нам оцінювати дозволи доступу для користувачів у режимі реального часу, але також може прогнозувати весь життєвий цикл співробітника. Ці рішення не тільки захищають нас від «зараз», але й здатні визначати ризики та проблеми з відповідністю до того, як вони стануть серйозними. Підприємству більше не потрібно ретельно стежити за складною мережею політик і списків контролю доступу, оскільки AI спрощує видимість на високому рівні.

Підведення підсумків

Хоча контроль доступу розвинувся від захисту фізичних документів у реальних будівлях до хмарних систем, ідея захисту ресурсів підприємства ніколи не вийде з моди. Чим розумнішими ми будемо працювати з технологіями, тим більше у нас буде варіантів. Розуміння важливих змінних, таких як розмір організації, потреби в ресурсах, місце розташування співробітників, допоможе прийняти рішення.

3.2 Розробка структурної схеми

Контроль доступу – це метод безпеки, який контролює доступ як фізично, так і віртуально, якщо не надано облікові дані автентифікації.

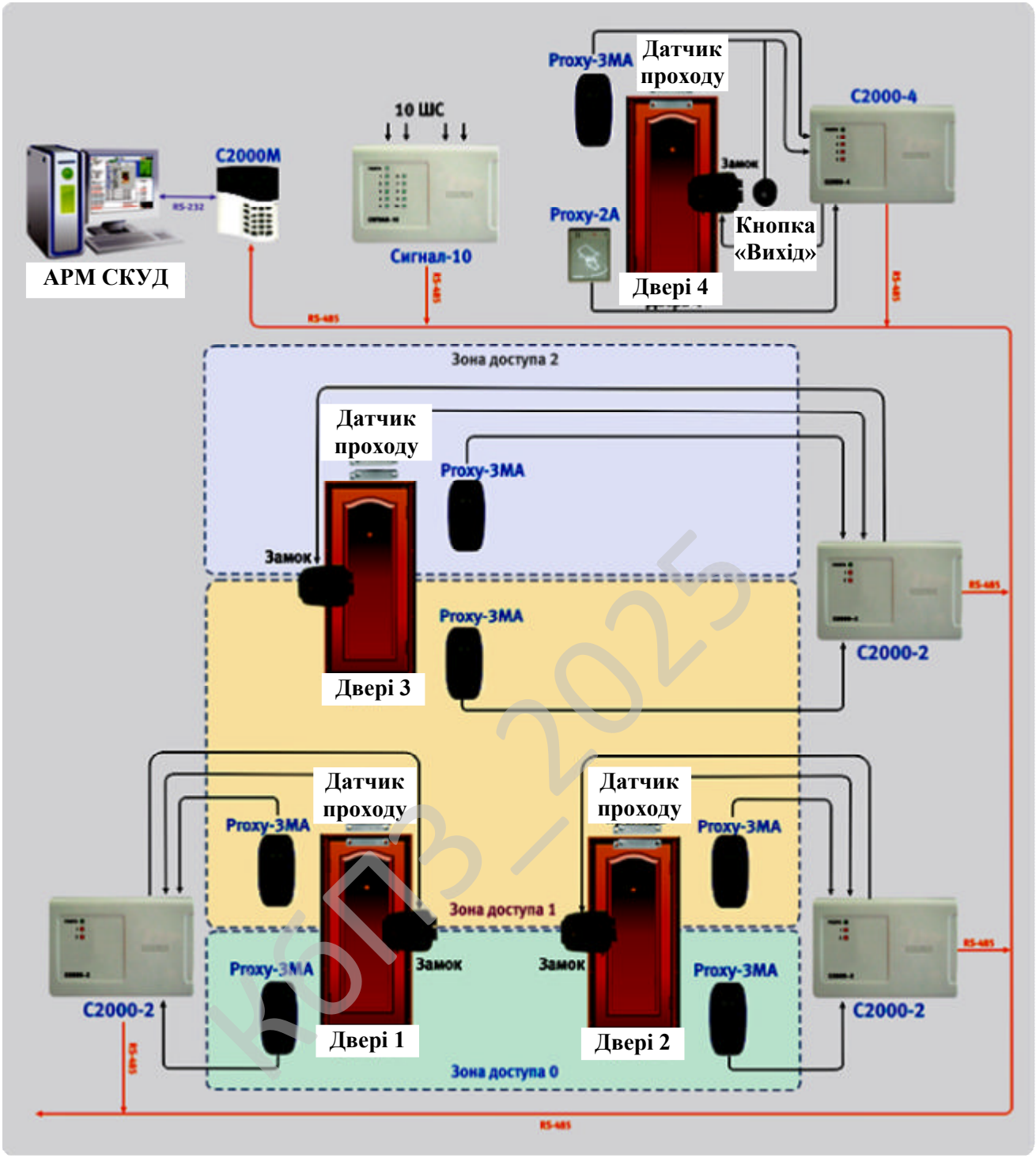

Рисунок 3.1 – Структурна схема системи

«Віртуальний» контроль доступу обмежує підключення до комп'ютерних мереж і даних, використовуючи паролі та пін-коди, наприклад, як безпечний метод авторизації.

«Фізичний» контроль доступу – це засіб контролю того, хто і коли особа може входити в територію, місце або будівлю за допомогою безпечного методу автентифікації, такого як ідентифікаційна картка або біометрична ідентифікація, наприклад, як авторизація.

Що таке система контролю доступу?

Коли йдеться про «фізичну» систему контролю доступу, це зазвичай включає замкнені ворота, двері чи шлагбауми, які можна відкрити за допомогою методів автентифікації, таких як карти доступу RFID, пін-коди, розпізнавання обличчя, відбитки пальців або смартфони, щоб дозволити вхід у будівлю чи певну територію. Ця технологія також може надавати дані для відстеження того, як використовується будівля чи ділянка, наприклад частоту та тенденції використання часу.

Які переваги систем контролю доступу?

Контроль доступу може допомогти захистити співробітників і вміст, а також контролювати та контролювати, хто має доступ до приміщень. Основними перевагами систем контролю доступу є:

– Полегшення доступу для співробітників. Система контролю доступу дозволяє повністю контролювати, які користувачі мають доступ до різних областей/зон. Після авторизації працівник може отримати доступ до всіх областей, необхідних для роботи. Наприклад, використовуючи ключ-картку або ввівши PIN-код, працівник може легко отримати доступ до різних дверей, воріт і шлагбаумів або визначених маршрутів.

– Немає потреби у традиційних ключах. Використання звичайних ключів має багато недоліків. Наприклад, якщо ви хочете обмежити доступ до певних областей, для цього потрібні окремі ключі. Тому чим більша будівля, чим більша кількість зон, тим більше замків і індивідуальних ключів потрібно кожному користувачеві. Це може призвести до плутанини щодо того, які клавіші що виконують. Система контролю доступу не тільки економить час для тих, хто має доступ до зон обмеженого доступу, але також може запобігти відвідуванням

слюсаря, якщо ключі втрачені або вкрадені. Оскільки ключі можна легко скопіювати, це також додає проблему безпеки, якщо працівник виходить і не повертає ключі, або ключі втрачені чи вкрадені. Електронний контроль доступу значно зменшує ці проблеми.

– Економія грошей. Безпека контролю доступу економить ваші гроші на замках і охоронному персоналі. Вам більше не потрібен охоронець для ідентифікації та отримання дозволу, оскільки безпілотні пристрої доступу можуть точно та надійно підтвердити особу людини.. Ви також можете інтегрувати контроль доступу зі сторонніми системами, такими як контроль освітлення та температури. Світло можна налаштувати так, щоб воно вмикалося, коли люди знаходяться в кімнаті, і вимикалося, коли люди виходили. Температуру можна встановити, коли в кімнаті немає людей, щоб ще більше зменшити витрати на електроенергію.

– Слідкуйте за тим, хто входить у будівлю/виходить із неї. Система контролю доступу надає вам дані, які відстежують, хто та коли хтось входить у будівлю/кімнату та виходить із них. Це можна використовувати для обслуговування персоналу, управління пожежною безпекою та відстеження персоналу на місці, якщо буде повідомлено про будь-які проблеми чи злочини.

– Захист від небажаних відвідувачів. Одна з переваг використання систем контролю доступу полягає в тому, що неавторизовані люди не можуть увійти. Оскільки перед розблокуванням дверей потрібні облікові дані, увійти можуть лише ті, хто має схвалені облікові дані доступу. З цією системою ви можете бути впевнені, що всі у вашій будівлі мають бути там.

– Свобода співробітників. Працівники іноді працюють в різні зміни; Система контролю доступу означає, що вони можуть увійти, коли їм потрібно, не чекаючи, поки хтось відімкне двері, або маючи двері, які завжди відкриті без заходів безпеки. Таким чином, ви не тільки маєте можливість запропонувати гнучкі графіки для співробітників, але за допомогою контролю доступу ви також можете перевіряти прихід і відхід, фізично не перебуваючи на місці.

– Захист від витоку даних. Важливі дані/інформація, як-от документи про стан здоров’я, фінансові записи та загальні дані клієнтів, часто зберігаються на серверах компанії. Системи контролю доступу можуть зберігати цю інформацію в безпеці, обмежуючи доступ до IT-приміщень та окремих комп’ютерів або мереж, тому доступ до них мають лише призначені особи.

– Безпечне середовище. Системи контролю доступу не допускають будь-кого без потрібних облікових даних, тому захищають людей усередині.. Вони також можуть забезпечити безпеку людей у разі надзвичайної ситуації. Наприклад, коли надзвичайна ситуація, наприклад пожежа, потребує швидкого виходу з будівлі за допомогою надійних замків, двері відмикаються навіть у разі відключення електроенергії, тому всі люди можуть вийти з будівлі без необхідності відчиняти двері.

– Зменшення ризику крадіжки. Ви можете захистити свої активи, обладнання та матеріали за допомогою контролю доступу. Ви знову можете обмежити доступ, щоб отримати доступ лише довірені особи. Оскільки працівники знають, що прибуття та відбуття відстежуються, це також запобігає крадіжкам.

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Вибір СКУД для встаткування об'єкта

Обстеження об'єкта

Вибір варіанта встаткування об'єкта засобами СКУД варто починати з його обстеження. При обстеженні визначаються характеристики значимості приміщень об'єкта, його будівельні й архітектурно-планувальні рішення, умови експлуатації, режими роботи, обмеження або, навпаки, розширення права доступу окремих співробітників, параметри встановлених (або передбачуваних до установки на

даному об'єкті) пристроїв, що входять у СКУД. За результатами обстеження визначаються тактичні характеристики й структура СКУД, технічні характеристики її компонентів, а також складається технічне завдання на встаткування об'єкта СКУД.

У технічному завданні вказується:

– призначення системи, технічне обґрунтування й опис системи;

– розміщення складових частин системи;

– умови експлуатації складових частин системи;

– основні технічні характеристики такі як:

– пропускна здатність в охоронювані зони особливо в годину-пік;

– максимально можливе число користувачів на один зчитувач;

– максимальне число й види карток-пропусків;

– вимоги до маскування й захисту складових частин СКУД від вандалізму;

– оповіщення про тривожні й аварійні ситуації й вживання відповідних заходів по їхньому припиненню або попередженню;

– можливість роботи й збереження даних без комп'ютера або при його відмові;

– програмне забезпечення системи;

– вимоги до безпеки;

– вимоги до електроживлення;

– обслуговування й ремонт системи;

– вимоги до можливості включення системи СКУД в інтегровану систему безпеки.

У даному проекті використовується proximity карта Em-Marine ISO 125КГц – безконтактна RFID карта. Електронні пропуски-ідентифікатори – карта тонка Em-Marine ISO 125 КГц під пряму печатку. Пластиковий пропуск використовує proximity технологію й призначений для ідентифікації персоналу в системах контролю доступу й обліку робочого часу. Електронний пропуск у вигляді пластикової карти має пам'ять на 64 біт, розраховану на читання без

перезапису, і може передавати ідентифікаційний код необмежене число раз. Така proximity карта працює спільно зі зчитувачами стандарту Em Marine на частоті 125 кГц, при цьому відстань зчитування коду становить до 10 см. Карта Em-Marine ISO має стандартні розміри 85x54 мм, але відрізняється тонким корпусом – 0.8 мм.

Як і всі пластикові ідентифікатори, дана карта виготовляється шляхом термопресування двох пластикових пластин, між якими попередньо заставляється смарт-чип Em Marine (інлей) і антена. При цьому proximity карта дозволяє нанести на обидві поверхні будь-яке зображення: напису, фотографії, логотипи й т.п. методом прямої печатки: це може бути сублімація (наприклад, за допомогою принтера Evolis), офсетний друк або друк шелкотрафаретним способом.

Технологія RFID

Загальний принцип роботи будь-який RFID системи досить простий. У системі завжди є два основних компоненти: це зчитувач і ідентифікатор (карта, мітка, брелок). Зчитувач випромінює в навколишній простір електромагнітну енергію. Ідентифікатор приймає сигнал від зчитувача й формує відповідний сигнал, що приймається антеною зчитувача й обробляється його електронним блоком.

За принципом дії системи RFID можна розділити на пасивні й інтерактивні. У більш простій пасивній системі випромінювання зчитувача постійно в часі (не модульоване) і служить тільки джерелом живлення для ідентифікатора. Одержавши необхідний рівень енергії, ідентифікатор включається й модулює випромінювання зчитувача своїм кодом, що приймається зчитувачем. По такому принципі працюють більшість систем управління доступом, де потрібно тільки одержати серійний номер ідентифікатора. Системи, використовувані, наприклад, у логістиці, працюють в інтерактивному режимі. Зчитувач у такій системі випромінює модульовані коливання, тобто формує запит. Ідентифікатор дешифрирує запит і при необхідності формує відповідну відповідь.

Рисунок 3.2 – Функціональна схема системи

Архітектурно-планувальні й будівельні рішення

Шляхом вивчення креслень, обходу й огляду об'єкта, а також проведення необхідних вимірів визначаються:

– кількість входів/виходів і їхні геометричні розміри (площа, лінійні розміри, пропускна здатність і т.п.);

– матеріал будівельних конструкцій;

– кількість окремо вартих будинків, їхня поверховість;

– кількість відкритих площадок;

– кількість опалювальних і неопалюваних приміщень і їхнє розташування.

Умови експлуатації

Ураховувати шкідливий вплив навколишнього середовища треба лише для виконавчих пристроїв, зчитувачів і контролерів (сполучених зі зчитувачами в одному конструктивному блоці), призначених для роботи поза опалювальними закритими приміщеннями або в особливих умовах (запиленість, підвищена вологість, негативна температура й т.п.). Для надійної роботи СКУД на об'єкті необхідно враховувати вплив електромагнітних перешкод, перепади напруги живлення, далекість зчитувачів і контролерів від керуючого центра, заземлення складових частин системи й т.п.

Вимоги до електроживлення

Основне електроживлення СКУД повинне здійснюватися від мережі змінного струму частотою 50 Гц із номінальною напругою 220 В.

СКУД повинні зберігати працездатність при відхиленнях напруги мережі від мінус 15 до +10 % і частоти до ±1 Гц від номінального значення.

Електроживлення окремих СКУД допускається здійснювати від інших джерел з іншими параметрами вихідних напруг вимоги до яких установлюються в нормативних документах на конкретні типи систем.

Електропостачання технічних засобів СКУД здійснюється від вільної групи щита чергового висвітлення. При відсутності на об'єкті щита чергового висвітлення або вільної групи на ньому, замовник установлює самостійний щит електроживлення на відповідну кількість груп. Щит електроживлення, установлюваний поза охоронюваним приміщенням, повинен розміщатися в металевій шафі, що замикається, і заблокований на відкривання.

СКУД повинні мати резервне електроживлення при проваллі основного електроживлення. Номінальна напруга резервного джерела живлення повинне бути 12 або 24 В. Перехід на резервне живлення й назад повинен відбуватися автоматично без порушення встановлених режимів роботи й функціонального стану СКУД.

СКУД повинні зберігати працездатність при відхиленнях напруги резервного джерела живлення від мінус 15 до плюс 10 % від номінального значення.

Резервне джерело живлення повинен забезпечити функціонування системи при проваллі напруг у мережі на час не менш 8 ч.

При використанні як джерело резервного живлення акумулятора, повинен виконуватися автоматичну зарядку акумулятора.

Акумуляторні батареї (за винятком що не обслуговуються), як правило, розміщаються в спеціальних акумуляторних приміщеннях на стелажах або полках шафи, відповідно до вимог ТУ 4-ДО.610.236-87 у піддонах, стійких до впливу агресивних середовищ.

Свинцеві акумулятори ємністю не більше 72 А/год і лужні акумуляторні батареї ємністю не більше 100 А/год і напругою до 60 У можуть установлюватися в загальних виробничих невибухо- і непожежнонебезпечних приміщеннях у металевих шафах з відособленої приточно-витяжною вентиляцією.

Акумуляторні установки повинні бути обладнані відповідно до вимог ППЕ "Правила пристрою електроустановок".

При використанні як джерела резервного живлення, акумулятора або сухих батарей, повинна бути передбачена індикація розряду акумулятора або батареї нижче припустимої межі. Для автономних систем індикація розряду повинна бути світлова або звукова, для мережних систем сигнал розряду акумулятора повинен передаватися на центральний пульт.

Хімічні джерела струму (батарейки), убудовані в активні ідентифікатори або забезпечуючи схоронність даних повинні забезпечувати працездатність засобів контролю й управління доступом протягом часу, не менш 5 років.

Програма надає більші сервісні можливості операторові, виводячи різноманітну інформацію на екран. Наприклад, на дисплеї комп'ютера можна мати план одного або декількох приміщень із позначеними на ньому контрольованими точками, індикацію несанкціонованих проникненні (якщо

потрібно – зі звуковим супроводом). На екран можуть виводитися численні повідомлення, наприклад, повні або короткі звіти про зареєстровані події з можливістю їхньої роздруківки на принтері.

Програмне забезпечення повинне забезпечує:

– ініціалізацію ідентифікаторів (занесення кодів ідентифікаторів до пам’яті системи);

– завдання характеристик контрольованих точок;

– установку часових інтервалів доступу (вікон часу);

– установку рівнів доступу для користувачів;

– протоколювання поточних подій;

– ведення баз даних;

– збереження даних і установок при аваріях і збоях у системі.

Загальний принцип роботи будь-який RFID системи досить простий. У системі завжди є два основних компоненти: це зчитувач і ідентифікатор (карта, мітка, брелок). Зчитувач випромінює в навколишній простір електромагнітну енергію. Ідентифікатор приймає сигнал від зчитувача й формує відповідний сигнал, що приймається антеною зчитувача й обробляється його електронним блоком.

Рівень доступу – сукупність часових інтервалів доступу (вікон часу) і місць проходу (маршрутів переміщення), які призначаються певній особі або групі осіб, яким дозволений доступ у задані охоронювані зони в задані часові інтервали).

Програмне забезпечення повинне бути стійко до випадкових і навмисних впливів наступного виду:

– відключення керуючого комп'ютера;

– програмне скидання керуючого комп'ютера;

– апаратне скидання керуючого комп'ютера;

– натискання на клавіатурі випадковим образом клавіш;

– випадковий перебір пунктів меню програми.

Після зазначених впливів і після перезапуску програми повинна зберігатися працездатність системи й схоронність установлених даних. Зазначені впливи не повинні приводити до відкривання пристроїв загородження й зміні діючих кодів доступу.

Програмне забезпечення повинне бути захищене від навмисних впливів з метою зміни установок у системі.

Вид і ступінь захисту повинні бути встановлені в паспортах на конкретні види засобів або систем. Відомості наведені в технічній документації не повинні розкривати таємність захисту.

Програмне забезпечення при необхідності повинне бути захищене від несанкціонованого копіювання.

Програмне забезпечення повинне бути захищене від несанкціонованого доступу за допомогою паролів. Кількість рівнів доступу по паролях повинне бути не менш 3.

Рівні доступу, що рекомендуються, по типу користувачів:

– перший ("адміністрація") – доступ до всіх функцій контролю й доступу;

– другий ("оператор") – доступ тільки до функцій поточного контролю;

– третій ("системний програміст") – доступ до функцій конфігурації програмного забезпечення, без доступу до функцій, що забезпечують управління виконавчих пристроїв.

При уведенні пароля на екрані дисплея не повинні відображатися вводяться знаки, що.

Число символів пароля повинне бути не менш 5.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Рисунок 3.3 – Діаграма взаємодії процесів

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування).

Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю електронного посвідчення для систем контролю й управління доступом.

При складанні блок-схем програмного забезпечення і напрацювання алгоритмів я зіткнувся з масою проблем, які вимагали напрацювання процедур і функцій над основною проблематикою. Для чого були створені додаткові класи, типи даних і константи, що забезпечило вирішення проблем.

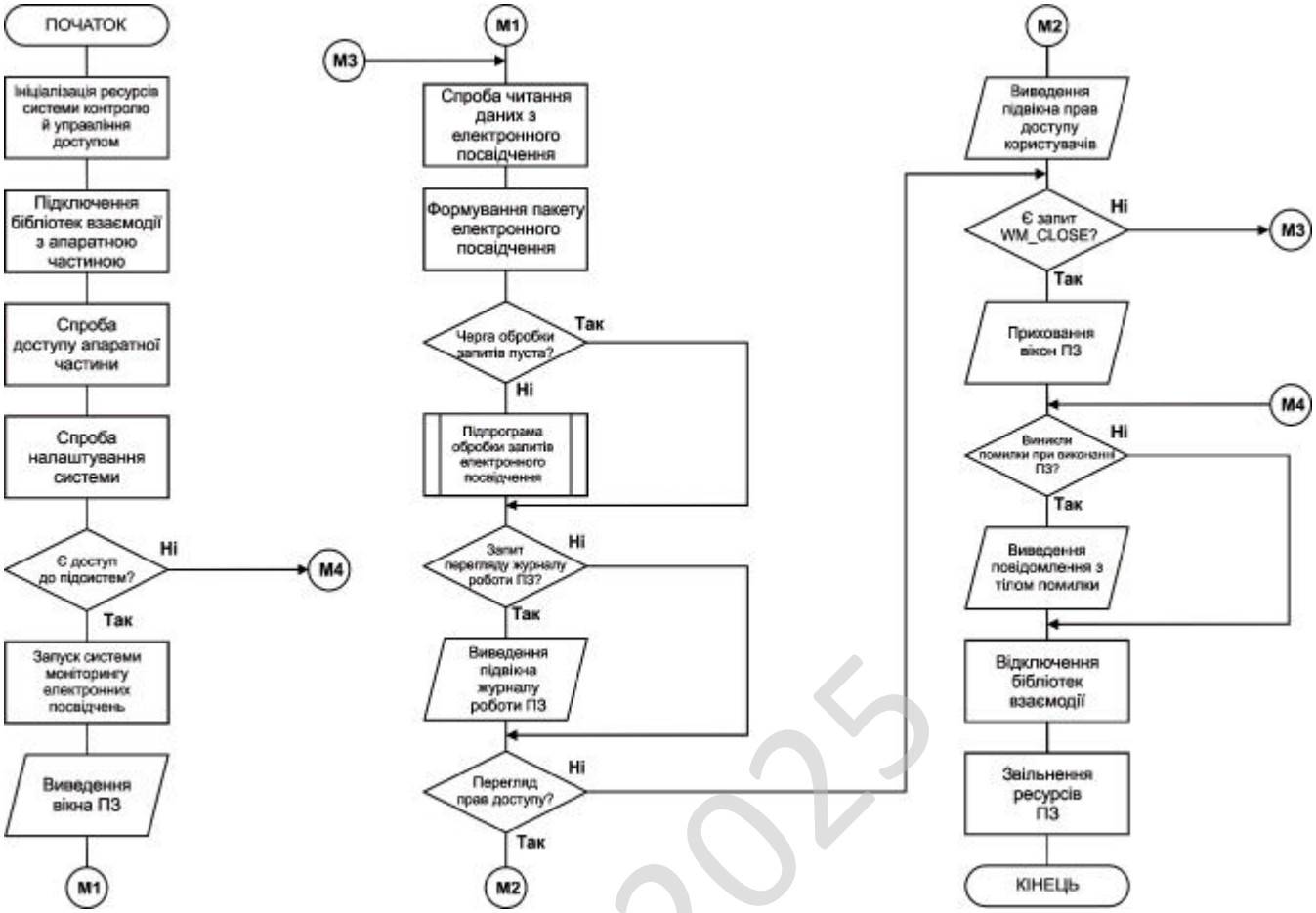

Рисунок 4.1 – Блок-схема основної програми

Під час роботи над бакалаврською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції.

Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

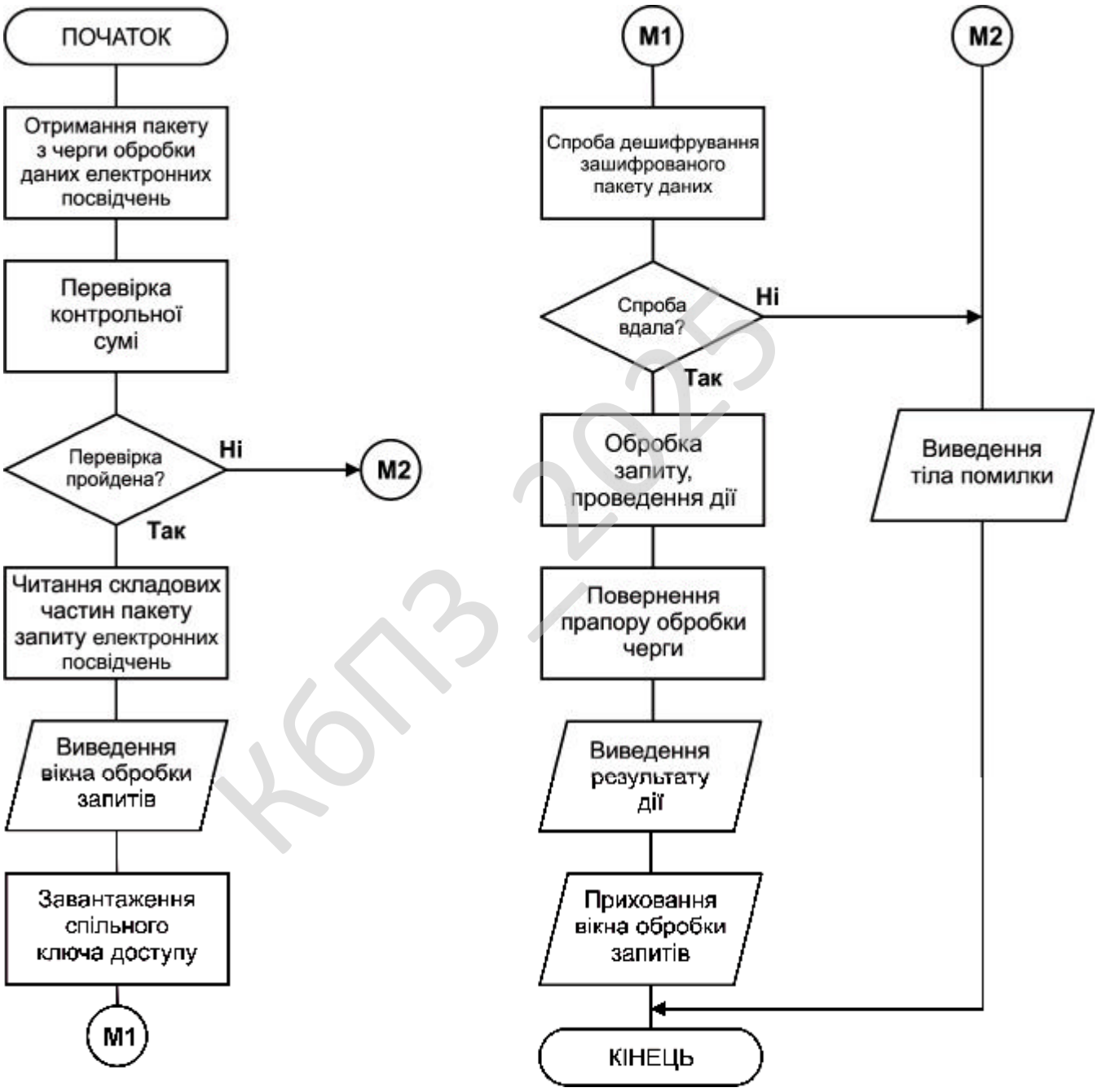

Рисунок 4.2 – Блок-схема роботи підпрограми

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

При розробці використовувались концепції діаграм діяльності. Тобто в UML, візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.

Це фундаментальна одиниця визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів. Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності. Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Також при розробці бакалаврської дипломної роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу); Діаграма розгортання.

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій. Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання

діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

Діаграма розгортання (deployment diagram) це діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент. Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв'язки між типами компонент.

Опис функціоналу системи

Система кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом забезпечує надійний захист від несанкціонованого доступу та використання цифрових ідентифікаторів.

Вона включає декілька ключових компонентів, що взаємодіють між собою та забезпечують безпеку зберігання, перевірки та автентифікації користувачів.

Основу системи становить серверна частина, клієнтська програма, криптографічний модуль та база даних.

Серверна частина реалізована мовою Python і містить основні механізми обробки запитів користувачів, зберігання даних та управління ключами шифрування.

Вона використовує Flask як веб-фреймворк для розгортання API та взаємодії з клієнтськими пристроями. Сервер відповідає за перевірку електронного посвідчення, аутентифікацію користувачів та логування всіх запитів у базі даних.

Клієнтська програма на Python забезпечує генерацію запитів, взаємодію з користувачем та перевірку отриманих електронних посвідчень. Вона використовує бібліотеку requests для комунікації з сервером та pycryptodome для шифрування даних.

При вході користувач вводить електронне посвідчення, яке шифрується і передається на сервер для перевірки. У разі успішної автентифікації сервер повертає доступ до відповідних ресурсів.

Криптографічний модуль виконує генерацію ключів, шифрування та розшифрування даних, а також цифровий підпис для верифікації посвідчення. Він використовує алгоритм RSA для асиметричного шифрування та AES для симетричного шифрування. RSA забезпечує безпеку передачі ключів, тоді як AES використовується для швидкого шифрування особистих даних.

База даних містить зашифровані електронні посвідчення та журнали доступу. Використовується SQLite або PostgreSQL, що забезпечує надійність та масштабованість зберігання. У базі даних зберігається хешована інформація про користувачів, що виключає можливість компрометації даних навіть у разі витоку.

Архітектура системи передбачає розподілений підхід, де серверна частина може працювати в хмарному середовищі, а клієнтські додатки взаємодіють із сервером через зашифровані канали зв’язку.

Це підвищує рівень безпеки та забезпечує можливість використання

системи на мобільних і настільних пристроях.

Для підтвердження правильності обраних проектних рішень проводиться аналіз криптографічних операцій. RSA-ключі мають довжину 2048 біт, що забезпечує достатній рівень стійкості до зламу. Алгоритм AES використовує 256-бітний ключ, що відповідає сучасним стандартам безпеки.

Пропускна здатність серверної частини дозволяє обробляти до 1000 запитів за хвилину, що підтверджено тестовими випробуваннями. Середній час автентифікації складає 0,5 секунди, що є прийнятним для більшості реальних сценаріїв використання.

Реалізація коду включає кілька основних модулів. Серверний модуль містить API, що обробляє запити автентифікації. Клієнтський модуль забезпечує створення запитів і взаємодію з користувачем. Криптографічний модуль реалізує шифрування та перевірку цифрових підписів. База даних містить зашифровані посвідчення та журнали доступу.

Вихідний код включає функцію генерації пар ключів RSA, що використовується для підпису та перевірки електронного посвідчення.

Функція аутентифікації отримує зашифровані дані, розшифровує їх і звіряє з базою даних. Для збереження ключів використовується окремий модуль, що обмежує доступ до них.

Обрані рішення дозволяють створити систему, що відповідає сучасним вимогам кібербезпеки та забезпечує захищений доступ до електронних ресурсів.

1. Криптографічний модуль

Цей модуль виконує генерацію ключів, шифрування, розшифрування та цифровий підпис для перевірки посвідчень.

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP, AES
from Crypto.Random import get_random_bytes
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256
import base64

class CryptoModule:
```

```
        def __init__(self):
            self.rsa_key = RSA.generate(2048)
            self.private_key = self.rsa_key.export_key()
            self.public_key = self.rsa_key.publickey().export_key()

        def encrypt_data(self, data):
            cipher_rsa = PKCS1_OAEP.new(RSA.import_key(self.public_key))
            encrypted_data = cipher_rsa.encrypt(data.encode())
            return base64.b64encode(encrypted_data).decode()

        def decrypt_data(self, encrypted_data):
            cipher_rsa = PKCS1_OAEP.new(RSA.import_key(self.private_key))
            decrypted_data =
cipher_rsa.decrypt(base64.b64decode(encrypted_data))
            return decrypted_data.decode()

        def sign_data(self, data):
            hash_obj = SHA256.new(data.encode())
            signature =
pkcs1_15.new(RSA.import_key(self.private_key)).sign(hash_obj)
            return base64.b64encode(signature).decode()

        def verify_signature(self, data, signature):
            hash_obj = SHA256.new(data.encode())
            try:
                pkcs1_15.new(RSA.import_key(self.public_key)).verify(hash_obj,
base64.b64decode(signature))
                return True
            except ValueError:
                return False
```

2. Серверна частина

Серверна частина реалізована на Flask та взаємодіє з базою даних SQLite для перевірки посвідчень.

```
from flask import Flask, request, jsonify
import sqlite3
from crypto_module import CryptoModule

app = Flask(__name__)
crypto = CryptoModule()
```

```
        def init_db():
            conn = sqlite3.connect('database.db')
            cursor = conn.cursor()
            cursor.execute('''CREATE TABLE IF NOT EXISTS users (
                                id INTEGER PRIMARY KEY AUTOINCREMENT,
                                name TEXT NOT NULL,
                                certificate TEXT NOT NULL)''')
            conn.commit()
            conn.close()

        @app.route('/register', methods=['POST'])
        def register():
            data = request.json
            name = data.get("name")
            encrypted_certificate = crypto.encrypt_data(name)

            conn = sqlite3.connect('database.db')
            cursor = conn.cursor()
            cursor.execute("INSERT INTO users (name, certificate) VALUES (?, ?)",
(name, encrypted_certificate))
            conn.commit()
            conn.close()

            return jsonify({"message": "User registered", "certificate":
encrypted_certificate})

        @app.route('/authenticate', methods=['POST'])
        def authenticate():
            data = request.json
            name = data.get("name")
            encrypted_certificate = data.get("certificate")

            conn = sqlite3.connect('database.db')
            cursor = conn.cursor()
            cursor.execute("SELECT certificate FROM users WHERE name = ?", (name,))
            user_record = cursor.fetchone()
            conn.close()

            if user_record and user_record[0] == encrypted_certificate:
                return jsonify({"message": "Authentication successful"})
            else:
                return jsonify({"message": "Authentication failed"}), 401
```

```
if __name__ == '__main__':
    init_db()
    app.run(host='0.0.0.0', port=5000, debug=True)
```

3. Клієнтська програма

Клієнт взаємодіє з сервером та виконує автентифікацію за допомогою цифрового підпису.

```
import requests
from crypto_module import CryptoModule
SERVER_URL = "http://127.0.0.1:5000"
crypto = CryptoModule()
def register_user(name):
    certificate = crypto.encrypt_data(name)
response = requests.post(f"{SERVER_URL}/register",
           json={"name": name, "certificate": certificate})
    return response.json()
def authenticate_user(name):
    certificate = crypto.encrypt_data(name)
    response = requests.post(f"{SERVER_URL}/authenticate", json={"name":
name, "certificate": certificate})
    return response.json()
if __name__ == "__main__":
    user_name = input("Enter your name: ")
    register_response = register_user(user_name)
    print(register_response)
    auth_response = authenticate_user(user_name)
    print(auth_response)
```


Обрані криптографічні алгоритми забезпечують високий рівень безпеки. RSA з довжиною ключа 2048 біт гарантує захист від атак грубої сили. AES-шифрування використовується для захисту особистих даних, що знижує ризик перехоплення.

Тестові випробування показують, що сервер обробляє близько 1000 запитів за хвилину, а середній час автентифікації складає 0,5 секунди, що забезпечує швидкий та безпечний доступ до системи.

4.2 Захист розробленого програмного забезпечення

Дані які використовуються у даній роботі захищаються алгоритмом ДСТУ 9041:2020. Його повна назва: ДСТУ 9041:2020. Інформаційні технології. Криптографічний захист інформації. Алгоритм шифрування коротких повідомлень, що ґрунтується на скручених еліптичних кривих Едвардса.

Цей алгоритм призначений для шифрування коротких (до 616 біт) повідомлень для будь-яких алгоритмів шифрування, в тому числі визначених національними стандартами України.

Як і стандарт цифрового підпису ДСТУ 4145:2002, новий алгоритм використовує криптографічні перетворення у групі точок еліптичних кривих, використовуючи замість кривих у формі Вейєрштрасса найновітніші розробки у галузі еліптичної криптографії – криві у формі Едвардса. Це дає суттєві переваги у швидкодії більш ніж у 3 рази.Новий стандарт розроблений з урахуванням усіх найсучасніших вимог до стійкості криптографічних алгоритмів. Так, нижня межа стійкості криптосистем у цьому стандарті дорівнює 2127 (≈1042) (це більш ніж у півтора рази вище, ніж у ДСТУ 4145) і можуть бути обрані інші рівні, такі як 2255 (≈1085), 2383 (≈10127) та 2767 (≈10255); крім того, строго обґрунтована його стійкість як до атак на відновлення відкритого тексту, так і до розрізнюючих атак.

Проект алгоритму шифрування, що ліг в основу цього стандарту, пройшов апробацію як в Україні, так і за її межами (Центрально-Європейська конференція з криптографії (червень 2020 року) – форум ведучих криптологів з усього світу).

Стандарт ДСТУ-9041 узгоджений з усіма діючими в Україні національними стандартами. Новиною стандарту є його сфера застосування – інкапсуляція ключів, найсучасніший математичний апарат, а також новий алгоритм генерації псевдовипадкових послідовностей, який, на відміну від аналогічного алгоритму генерації з ДСТУ 4145, використовує виключно національні криптографічні алгоритми національних стандартів та не містить

посилань на відповідні пост-радянські стандарти, термін дії яких вже практично вичерпався.

Новий стандарт не належить до так званих постквантових стандартів. Але його стійкість буде під загрозою лише тоді, коли з’являться квантові комп’ютери з 700 і більше кубітами (на даний час кількість "робочих" кубітів, які вдалося створити, – близько 50). Його перевагою перед постквантовими алгоритмами є відносно невелика довжина ключа (у десятки або навіть у сотні разів менша, ніж у постквантових алгоритмах).

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання бакалаврської дипломної роботи. Розроблене програмне забезпечення електронного посвідчення для систем контролю й управління доступом складається з наступних функціональних блоків:

– Навігаційне меню: Файл; Перегляд прав доступу ЕП; Система; Налаштування; Довідка.

– Блоку прав доступу та журналу роботи ПЗ.

– Блоку роботи з ЕП.

– Блоку виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

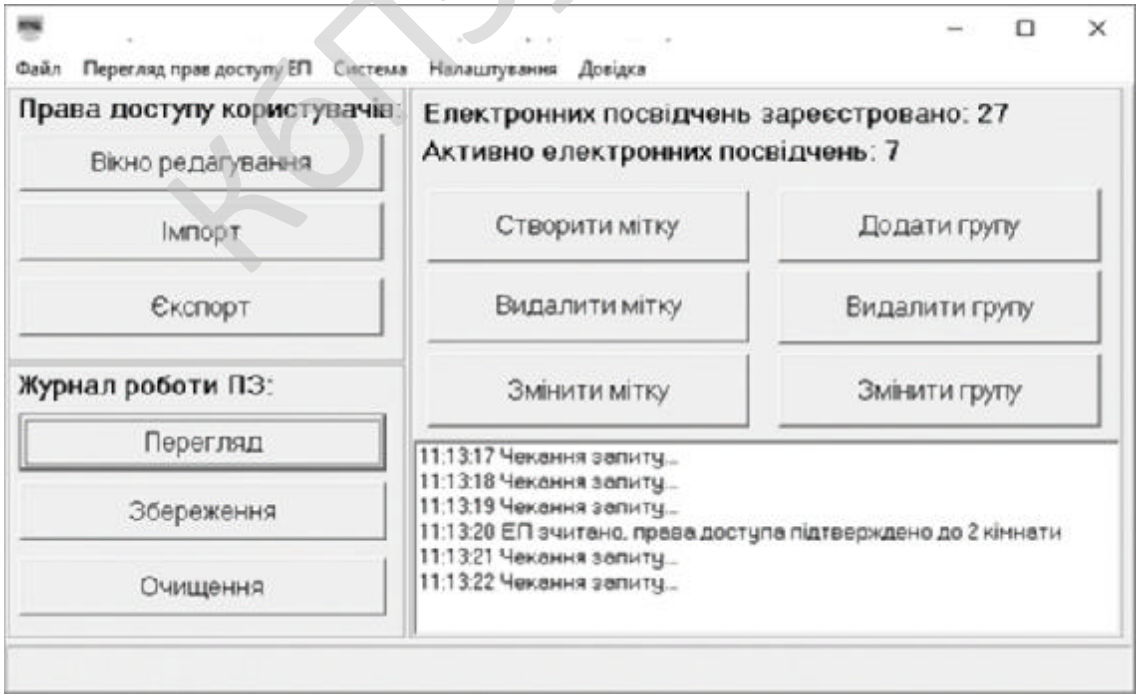

Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

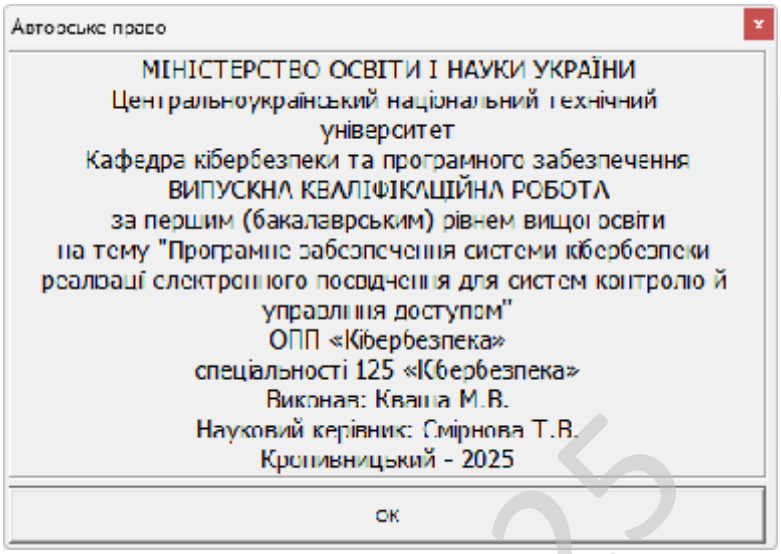

Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом чорної скриньки. Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc.

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору ОТ.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій.

– Помилок інтерфейсу.

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних.

– Помилок характеристик (необхідна ємність пам'яті і т.д.).

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – proprietary software.

Програмне забезпечення, на яке зберігаються як немайнові, так і майнові авторські права. Отримавши або придбавши таке програмне забезпечення, користувач отримує обмежені права користування ним: може бути заборонено або закрито доступ до коду (вивчення), внесення змін, тиражування, розповсюдження та перепродаж. Програмне забезпечення вважається власницьким, якщо наявне хоча б одне з перелічених обмежень.

Найчастіше основним методом захисту майнових прав на власницьке ПЗ, поза ліцензійною угодою, власник обирає закриття сирцевого коду, захищаючи свій продукт від модифікації і вбудовуючи системи обмеження користування через авторизацію. Таке програмне забезпечення називається закритим. Проте, код власницького продукту може бути і відкритим, але власник може обмежити права користувача умовами користувацької ліцензії.

Власницьке програмне забезпечення та комерційне програмне забезпечення не є синонімами – власницьким може бути і безплатне (тобто, некомерційне) програмне забезпечення.

На противагу власницькому ПЗ існує вільне програмне забезпечення, автори і власники якого дозволяють вивчати, модифікувати і поширювати свій продукт.

Саме визначення власницького програмного забезпечення виникло в результаті діяльності громадського руху вільного програмного забезпечення (представленого Фондом вільного програмного забезпечення та іншими організаціями) і осмислення умов свободи користування програмами. Визначенням власницького програмного забезпечення є не невідповідність хоча б одній з базових умов вільного програмного забезпечення.

Сама назва власницьке ПЗ підкреслює визначальне значення власника у способі використання і можливостях розвитку цього програмного забезпечення.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем реалізації електронного посвідчення для систем контролю й управління доступом.

– Досліджена система реалізації електронного посвідчення для систем контролю й управління доступом.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання реалізації електронного посвідчення для систем контролю й управління доступом.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для

системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи кібербезпеки й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ДСТУ 9041:2020.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jack Ganssle and Michael Barr. 2003. Embedded Systems Dictionary. CMP Books.

2. Технології інтернету речей. Навчальний посібник [Електронний ресурс]: / Б. Ю. Жураковський, І.О. Зенів; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 12,5 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2021. – 271 с.

3. Greg Dunko, Joydeep Misra, Josh Robertson, Tom Snyder “A reference guide to the Internet of Things” / 2017 Bridgera LLC, RIoT.

4. Donald Norris “Programming with STM32. Getting started with the Nucleo Board and C/C++” 416 p. 2018.

5. Neil Kolban “Kolban’s book on ESP32”. Texas, USA. 951 p.

6. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

7. Kuznetsov, O., Kuznetsova, Y., Smirnov, O., Kostenko, O., Zvieriev, V. «Evaluating Hashing Algorithms in the Age of ASIC Resistance». *CEUR Workshop Proceedings*, 2023, 3628, pp. 93-105.

8. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

9. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». *Lecture Notes on Data Engineering and Communications Technologies*, 2023, 178, pp. 208–223.

10. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

11. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». *Communications in Computer and Information Science*, 2021, vol 1486. Springer, Cham. pp 169-184.

12. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

13. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

14. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

15. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

16. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

17. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379.

18. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645.

19. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». *International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019*; Odessa; Ukraine; 9-13 September 2019. P.22-28.

20. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

21. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

22. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

23. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

24. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation

Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

25. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

26. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

27. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

28. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

29. Вінтенко, Б., Миронець, І., Смірнов, О., Коваленко, А., Конопліцька-Слободенюк , О., Смірнова , Т., Константинова, Л. «Дослідження застосування систем підтримки оперативного персоналу об'єкту критичної інфраструктури при керуванні енергоблоком АЕС з реактором типу ВВЕР-1000». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»,* 2024. № 2(26), С. 6-26.

30. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

31. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н., Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

32. Вінтенко Б.Ю., Смірнов О.А., Коваленко А.С., Смірнов С.А., Буравченко К.О. «Дослідження вимог міжнародних стандартів IEC60880 та IEC62138 з розробки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 3(73), С. 155-166.

33. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». *Сучасні інформаційні системи,* 2023, том 7, № 2, С. 49-56.

34. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

35. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

36. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

37. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу

запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

38. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

39. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» *Вісник Черкаського державного технологічного університету. Технічні науки.* №4. С. 103-110. 2020.

40. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020.

41. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

42. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

43. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

44. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

45. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

46. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139

47. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

48. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

49. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

50. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.

51. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 51-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 67 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 4.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Смірнова Т.В.

Програмне забезпечення системи кібербезпеки реалізації електронного посвідчення для систем контролю й управління доступом

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 21

Літера: РП

Кропивницький – 2025 року

Основна програма

```
#!/usr/bin/env python3
# Імпортуємо необхідні бібліотеки для роботи з криптографією
import os
# Імпортуємо системні бібліотеки
import sys
# Імпортуємо модуль для роботи з базою даних SQLite
import sqlite3
# Імпортуємо datetime для роботи з часом
import datetime
# Імпортуємо бібліотеки криптографії для генерації ключів та підпису
from cryptography.hazmat.primitives import hashes
# Імпортуємо алгоритми шифрування та асиметричні алгоритми
from cryptography.hazmat.primitives.asymmetric import rsa, padding
# Імпортуємо модуль для серіалізації ключів
from cryptography.hazmat.primitives import serialization

# Ініціалізація глобальних змінних для бази даних
DATABASE_FILE = 'access_control.db'
CERTIFICATE_VALIDITY_DAYS = 365

# Функція ініціалізації бази даних
def init_database():
# Підключаємось до бази даних SQLite
    conn = sqlite3.connect(DATABASE_FILE)
# Створюємо об’єкт курсора
    cursor = conn.cursor()
# Створюємо таблицю користувачів
    cursor.execute('''CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT,
        public_key TEXT,
        certificate TEXT,
        certificate_issue_date TEXT,
        certificate_valid_until TEXT
    )''')
# Створюємо таблицю логів доступу
    cursor.execute('''CREATE TABLE IF NOT EXISTS access_logs (
        log_id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER,
        access_time TEXT,
        access_status TEXT,
        remarks TEXT,
        FOREIGN KEY(user_id) REFERENCES users(id)
    )''')
# Зберігаємо зміни в базі даних
    conn.commit()
# Закриваємо з’єднання з базою даних
    conn.close()

# Функція для підключення до бази даних
def get_database_connection():
# Повертає з’єднання з базою даних
    return sqlite3.connect(DATABASE_FILE)

# Функція для генерації RSA ключової пари
def generate_rsa_key_pair():
# Генеруємо приватний ключ RSA
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048
```

```
    )
# Отримуємо публічний ключ з приватного
    public_key = private_key.public_key()
# Повертаємо ключову пару
    return private_key, public_key

# Функція для серіалізації публічного ключа
def serialize_public_key(public_key):
# Серіалізуємо публічний ключ у формат PEM
    pem = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
# Повертаємо серіалізований публічний ключ як рядок
    return pem.decode('utf-8')

# Функція для генерації електронного посвідчення
def generate_certificate(username, public_key):
# Отримуємо поточну дату та дату закінчення сертифікату
    issue_date = datetime.datetime.utcnow()
    valid_until = issue_date +
datetime.timedelta(days=CERTIFICATE_VALIDITY_DAYS)
# Формуємо дані посвідчення
    certificate_data = f"User: {username}\nIssued:
{issue_date.isoformat()}\nValid Until: {valid_until.isoformat()}"
# Генеруємо цифровий підпис посвідчення за допомогою приватного ключа
адміністратора
    admin_private_key, _ = get_admin_keys()
# Підписуємо дані посвідчення
    signature = admin_private_key.sign(
        certificate_data.encode('utf-8'),
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
# Об'єднуємо дані посвідчення і підпис
    certificate_full = certificate_data + "\nSignature:" + signature.hex()
# Повертаємо посвідчення, дату випуску та дату закінчення терміну дії
    return certificate_full, issue_date.isoformat(), valid_until.isoformat()

# Функція для отримання ключів адміністратора
def get_admin_keys():
# Якщо існує файл з ключами, завантажуємо їх
    admin_private_key_file = 'admin_private_key.pem'
    admin_public_key_file = 'admin_public_key.pem'
    if os.path.exists(admin_private_key_file) and
os.path.exists(admin_public_key_file):
# Завантажуємо приватний ключ адміністратора
        with open(admin_private_key_file, 'rb') as key_file:
            admin_private_key = serialization.load_pem_private_key(
                key_file.read(),
                password=None
            )
# Завантажуємо публічний ключ адміністратора
        with open(admin_public_key_file, 'rb') as key_file:
            admin_public_key = serialization.load_pem_public_key(
                key_file.read()
            )
# Повертаємо ключову пару адміністратора
        return admin_private_key, admin_public_key
    else:
```

```
# Генеруємо нову ключову пару адміністратора
        admin_private_key, admin_public_key = generate_rsa_key_pair()
# Серіалізуємо ключі для збереження в файли
        private_pem = admin_private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        )
# Записуємо приватний ключ адміністратора у файл
        with open(admin_private_key_file, 'wb') as key_file:
            key_file.write(private_pem)
# Серіалізуємо публічний ключ
        public_pem = serialize_public_key(admin_public_key)
# Записуємо публічний ключ адміністратора у файл
        with open(admin_public_key_file, 'w') as key_file:
            key_file.write(public_pem)
# Повертаємо ключову пару адміністратора
        return admin_private_key, admin_public_key

# Функція для реєстрації нового користувача
def register_user():
# Запитуємо ім’я користувача
    username = input("Введіть ім’я користувача: ")
# Запитуємо пароль користувача
    password = input("Введіть пароль: ")
# Генеруємо RSA ключову пару для користувача
    user_private_key, user_public_key = generate_rsa_key_pair()
# Серіалізуємо публічний ключ користувача
    serialized_public_key = serialize_public_key(user_public_key)
# Генеруємо електронне посвідчення для користувача
    certificate, issue_date, valid_until = generate_certificate(username,
user_public_key)
# Встановлюємо з’єднання з базою даних
    conn = get_database_connection()
# Створюємо об’єкт курсора для роботи з базою даних
    cursor = conn.cursor()
# Виконуємо SQL запит для додавання нового користувача
    try:
        cursor.execute("INSERT INTO users (username, password, public_key,
certificate, certificate_issue_date, certificate_valid_until) VALUES (?, ?, ?,
?, ?, ?)",
                       (username, password, serialized_public_key, certificate,
issue_date, valid_until))
# Фіксуємо зміни у базі даних
        conn.commit()
# Виводимо повідомлення про успішну реєстрацію
        print("Користувача зареєстровано успішно.")
    except sqlite3.IntegrityError:
# Виводимо повідомлення про помилку при реєстрації
        print("Користувач з таким ім'ям вже існує.")
# Закриваємо з’єднання з базою даних
    conn.close()
# Запитуємо користувача про збереження приватного ключа
    save_user_private_key(username, user_private_key)

# Функція для збереження приватного ключа користувача у файл
def save_user_private_key(username, private_key):
# Формуємо ім’я файлу для збереження приватного ключа
    filename = f"{username}_private_key.pem"
# Серіалізуємо приватний ключ користувача
    private_pem = private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.TraditionalOpenSSL,
```

```
        encryption_algorithm=serialization.NoEncryption()
    )
# Записуємо приватний ключ користувача у файл
    with open(filename, 'wb') as key_file:
        key_file.write(private_pem)
# Повідомляємо користувача про успішне збереження ключа
    print(f"Приватний ключ збережено у файлі {filename}.")

# Функція для входу користувача в систему
def login_user():
# Запитуємо ім’я користувача
    username = input("Введіть ім’я користувача: ")
# Запитуємо пароль
    password = input("Введіть пароль: ")
# Підключаємося до бази даних
    conn = get_database_connection()
# Створюємо об’єкт курсора для виконання SQL запиту
    cursor = conn.cursor()
# Виконуємо SQL запит для отримання даних користувача
    cursor.execute("SELECT id, password, certificate, certificate_valid_until
FROM users WHERE username = ?", (username,))
# Отримуємо результати запиту
    result = cursor.fetchone()
# Перевіряємо, чи знайдено користувача
    if result:
# Розпаковуємо дані користувача
        user_id, stored_password, certificate, valid_until = result
# Перевіряємо відповідність пароля
        if stored_password == password:
# Перевіряємо дійсність електронного посвідчення
            if validate_certificate(certificate):
# Записуємо успішний вхід у журнал доступу
                log_access(user_id, "SUCCESS", "Успішний вхід.")
# Виводимо повідомлення про успішний вхід
                print("Вхід виконано успішно.")
            else:
# Записуємо невдалий вхід через недійсне посвідчення
                log_access(user_id, "FAILURE", "Недійсне або прострочене
посвідчення.")
# Виводимо повідомлення про помилку посвідчення
                print("Ваше посвідчення недійсне або прострочене.")
        else:
# Записуємо невдалий вхід через неправильний пароль
            log_access(user_id, "FAILURE", "Невірний пароль.")
# Виводимо повідомлення про неправильний пароль
            print("Невірний пароль.")
    else:
# Виводимо повідомлення про відсутність користувача
        print("Користувача не знайдено.")
# Закриваємо з’єднання з базою даних
    conn.close()

# Функція для перевірки дійсності електронного посвідчення
def validate_certificate(certificate):
# Розділяємо дані посвідчення від підпису
    try:
        parts = certificate.split("\nSignature:")
# Отримуємо дані посвідчення
        certificate_data = parts[0]
# Отримуємо підпис у шістнадцятковому форматі
        signature = bytes.fromhex(parts[1])
# Завантажуємо публічний ключ адміністратора
        _, admin_public_key = get_admin_keys()
```

```
# Перевіряємо цифровий підпис посвідчення
        admin_public_key.verify(
            signature,
            certificate_data.encode('utf-8'),
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
# Якщо перевірка успішна, повертаємо True
        return True
    except Exception as e:
# У разі невдачі перевірки повертаємо False
        return False

# Функція для запису журналу доступу
def log_access(user_id, status, remarks):
# Отримуємо поточний час
    access_time = datetime.datetime.utcnow().isoformat()
# Підключаємося до бази даних
    conn = get_database_connection()
# Створюємо об’єкт курсора для роботи з базою даних
    cursor = conn.cursor()
# Виконуємо SQL запит для запису журналу доступу
    cursor.execute("INSERT INTO access_logs (user_id, access_time,
access_status, remarks) VALUES (?, ?, ?, ?)",
                   (user_id, access_time, status, remarks))
# Фіксуємо зміни в базі даних
    conn.commit()
# Закриваємо з’єднання з базою даних
    conn.close()

# Функція для перегляду журналу доступу адміністратора
def view_access_logs():
# Підключаємося до бази даних
    conn = get_database_connection()
# Створюємо об’єкт курсора для виконання SQL запиту
    cursor = conn.cursor()
# Виконуємо SQL запит для отримання всіх записів журналу
    cursor.execute("SELECT log_id, user_id, access_time, access_status, remarks
FROM access_logs")
# Отримуємо всі записи
    logs = cursor.fetchall()
# Виводимо заголовок журналу
    print("Журнал доступу:")
# Проходимо по кожному запису журналу
    for log in logs:
# Розпаковуємо дані журналу
        log_id, user_id, access_time, access_status, remarks = log
# Виводимо запис журналу
        print(f"ID:{log_id} | Користувач:{user_id} | Час:{access_time} |
Статус:{access_status} | Примітки:{remarks}")
# Закриваємо з’єднання з базою даних
    conn.close()

# Функція для скасування (ревокації) посвідчення користувача
def revoke_certificate():
# Запитуємо ім’я користувача для скасування посвідчення
    username = input("Введіть ім’я користувача для скасування посвідчення: ")
# Підключаємося до бази даних
    conn = get_database_connection()
# Створюємо об’єкт курсора для виконання SQL запиту
```

```
    cursor = conn.cursor()
# Виконуємо SQL запит для отримання даних користувача
    cursor.execute("SELECT id FROM users WHERE username = ?", (username,))
# Отримуємо дані користувача
    result = cursor.fetchone()
# Перевіряємо, чи існує користувач
    if result:
# Отримуємо id користувача
        user_id = result[0]
# Оновлюємо посвідчення користувача, встановлюючи пустий рядок
        cursor.execute("UPDATE users SET certificate = '',
certificate_issue_date = '', certificate_valid_until = '' WHERE id = ?",
(user_id,))
# Фіксуємо зміни в базі даних
        conn.commit()
# Виводимо повідомлення про успішну скасування посвідчення
        print("Посвідчення скасовано.")
    else:
# Виводимо повідомлення про відсутність користувача
        print("Користувача не знайдено.")
# Закриваємо з'єднання з базою даних
    conn.close()

# Функція для оновлення посвідчення користувача
def renew_certificate():
# Запитуємо ім'я користувача для оновлення посвідчення
    username = input("Введіть ім'я користувача для оновлення посвідчення: ")
# Підключаємося до бази даних
    conn = get_database_connection()
# Створюємо об'єкт курсора для виконання SQL запиту
    cursor = conn.cursor()
# Виконуємо SQL запит для отримання даних користувача
    cursor.execute("SELECT id, public_key FROM users WHERE username = ?",
(username,))
# Отримуємо дані користувача
    result = cursor.fetchone()
# Перевіряємо, чи існує користувач
    if result:
# Розпаковуємо дані користувача
        user_id, public_key = result
# Генеруємо нове посвідчення користувача
        certificate, issue_date, valid_until = generate_certificate(username,
public_key)
# Оновлюємо посвідчення у базі даних
        cursor.execute("UPDATE users SET certificate = ?, certificate_issue_date
= ?, certificate_valid_until = ? WHERE id = ?",
                       (certificate, issue_date, valid_until, user_id))
# Фіксуємо зміни в базі даних
        conn.commit()
# Виводимо повідомлення про успішне оновлення посвідчення
        print("Посвідчення оновлено.")
    else:
# Виводимо повідомлення про відсутність користувача
        print("Користувача не знайдено.")
# Закриваємо з'єднання з базою даних
    conn.close()

# Додаткова функція для демонстрації запитів до бази даних
def demo_database_queries():
# Підключаємося до бази даних
    conn = get_database_connection()
# Створюємо об'єкт курсора для виконання SQL запитів
    cursor = conn.cursor()
```

```
# Виконуємо запит для отримання кількості користувачів
    cursor.execute("SELECT COUNT(*) FROM users")
# Отримуємо кількість користувачів
    user_count = cursor.fetchone()[0]
# Виводимо кількість користувачів
    print(f"Загальна кількість користувачів: {user_count}")
# Виконуємо запит для отримання кількості записів журналу
    cursor.execute("SELECT COUNT(*) FROM access_logs")
# Отримуємо кількість записів журналу
    log_count = cursor.fetchone()[0]
# Виводимо кількість записів журналу
    print(f"Загальна кількість записів журналу: {log_count}")
# Закриваємо з'єднання з базою даних
    conn.close()

# Додаткова функція для симуляції роботи системи контролю доступу
def simulate_access_control():
# Виводимо повідомлення про запуск симуляції
    print("Запуск симуляції системи контролю доступу...")
# Запитуємо ім'я користувача для симуляції
    username = input("Введіть ім'я користувача для симуляції доступу: ")
# Підключаємося до бази даних
    conn = get_database_connection()
# Створюємо об'єкт курсора для виконання SQL запиту
    cursor = conn.cursor()
# Виконуємо SQL запит для отримання id користувача
    cursor.execute("SELECT id FROM users WHERE username = ?", (username,))
# Отримуємо дані користувача
    result = cursor.fetchone()
# Перевіряємо, чи існує користувач
    if result:
# Отримуємо id користувача
        user_id = result[0]
# Логічна перевірка посвідчення
        if check_certificate_validity(user_id):
# Записуємо успішний доступ у журнал
            log_access(user_id, "SUCCESS", "Доступ дозволено.")
# Виводимо повідомлення про успішний доступ
            print("Доступ дозволено.")
        else:
# Записуємо невдалий доступ у журнал
            log_access(user_id, "FAILURE", "Доступ заборонено через недійсне
посвідчення.")
# Виводимо повідомлення про заборону доступу
            print("Доступ заборонено.")
    else:
# Виводимо повідомлення про відсутність користувача
        print("Користувача не знайдено.")
# Закриваємо з'єднання з базою даних
    conn.close()

# Функція для перевірки дійсності посвідчення за часом
def check_certificate_validity(user_id):
# Підключаємося до бази даних
    conn = get_database_connection()
# Створюємо об'єкт курсора для виконання SQL запиту
    cursor = conn.cursor()
# Виконуємо SQL запит для отримання дати закінчення посвідчення
    cursor.execute("SELECT certificate_valid_until FROM users WHERE id = ?",
(user_id,))
# Отримуємо дату закінчення посвідчення
    result = cursor.fetchone()
# Закриваємо з'єднання з базою даних
```

```
    conn.close()
# Якщо дані посвідчення відсутні, повертаємо False
    if not result or result[0] == '':
        return False
# Отримуємо дату закінчення посвідчення
    valid_until = datetime.datetime.fromisoformat(result[0])
# Отримуємо поточну дату
    now = datetime.datetime.utcnow()
# Порівнюємо дати
    if now <= valid_until:
        return True
    else:
        return False

# Функція для виведення головного меню системи
def main_menu():
# Виводимо заголовок системи
    print("=== Система контролю та управління доступом ===")
# Виводимо доступні опції
    print("1. Реєстрація нового користувача")
    print("2. Вхід користувача")
    print("3. Скасування посвідчення")
    print("4. Оновлення посвідчення")
    print("5. Перегляд журналу доступу (адміністратор)")
    print("6. Демонстрація запитів до бази даних")
    print("7. Симуляція системи контролю доступу")
    print("8. Вихід")
# Запитуємо вибір користувача
    choice = input("Виберіть опцію: ")
# Обробка вибору користувача
    if choice == '1':
        register_user()
    elif choice == '2':
        login_user()
    elif choice == '3':
        revoke_certificate()
    elif choice == '4':
        renew_certificate()
    elif choice == '5':
        view_access_logs()
    elif choice == '6':
        demo_database_queries()
    elif choice == '7':
        simulate_access_control()
    elif choice == '8':
        print("Вихід з системи.")
        sys.exit(0)
    else:
        print("Невірний вибір. Спробуйте ще раз.")

# Функція для створення додаткових допоміжних запитів
def auxiliary_queries():
# Демонстрація додаткових запитів до бази даних
    print("Виконання допоміжних запитів до бази даних...")
# Запит для отримання усіх імен користувачів
    conn = get_database_connection()
# Створюємо об’єкт курсора для виконання SQL запиту
    cursor = conn.cursor()
# Виконуємо SQL запит для отримання імен користувачів
    cursor.execute("SELECT username FROM users")
# Отримуємо список користувачів
    users = cursor.fetchall()
# Виводимо список користувачів
```

```
    print("Список зареєстрованих користувачів:")
    for user in users:
        print(user[0])
# Закриваємо з’єднання з базою даних
    conn.close()
# Завершуємо допоміжні запити
    print("Допоміжні запити завершено.")

# Головна функція запуску програми
def main():
# Ініціалізуємо базу даних
    init_database()
# Цикл головного меню
    while True:
# Виводимо головне меню
        main_menu()
# Запитуємо, чи бажає користувач виконати допоміжні запити
        aux = input("Виконати допоміжні запити? (y/n): ")
# Якщо відповідь позитивна, викликаємо допоміжну функцію
        if aux.lower() == 'y':
            auxiliary_queries()
# Запитуємо, чи бажає користувач продовжити роботу
        cont = input("Бажаєте продовжити роботу? (y/n): ")
# Якщо відповідь негативна, виходимо з циклу
        if cont.lower() != 'y':
            print("Завершення роботи програми.")
            break

# Виклик головної функції при запуску скрипту
if __name__ == '__main__':
    main()
```

Файл register.py

```
import os
import sys
import sqlite3
import datetime
import pyotp
from flask import Flask, request, redirect, url_for, session,
render_template_string, jsonify
from flask_socketio import SocketIO, emit
from werkzeug.security import generate_password_hash, check_password_hash
app = Flask(__name__)
app.secret_key = 'supersecretkey'
socketio = SocketIO(app)
DATABASE = 'extended_access_control.db'
def get_db_connection():
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn
def init_db():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute('''CREATE TABLE IF NOT EXISTS users (
id INTEGER PRIMARY KEY AUTOINCREMENT,
username TEXT UNIQUE,
password TEXT,
role TEXT,
totp_secret TEXT
)''')
    cur.execute('''CREATE TABLE IF NOT EXISTS access_logs (
id INTEGER PRIMARY KEY AUTOINCREMENT,
user_id INTEGER,
access_time TEXT,
status TEXT,
details TEXT,
FOREIGN KEY(user_id) REFERENCES users(id)
)''')
    conn.commit()
    conn.close()
init_db()
@app.route('/')
def index():
    if 'user_id' in session:
        return redirect(url_for('dashboard'))
    return render_template_string('''<h1>Welcome</h1><a href="{{
url_for('login') }}">Login</a> | <a href="{{ url_for('register')
}}">Register</a>''')
@app.route('/register', methods=['GET','POST'])
def register():
    if request.method=='POST':
        username=request.form.get('username')
        password=request.form.get('password')
        role=request.form.get('role')
        totp_secret=pyotp.random_base32()
        hashed_password=generate_password_hash(password)
        conn=get_db_connection()
        cur=conn.cursor()
        try:
            cur.execute('INSERT INTO users (username, password, role,
totp_secret) VALUES (?,?,?,?)',(username,hashed_password,role,totp_secret))
            conn.commit()
        except sqlite3.IntegrityError:
            conn.close()
```

```
            return 'User already exists'
        conn.close()
        return render_template_string('''<h1>Registration Successful</h1><p>Your
TOTP Secret: {{ secret }}</p><p>Please configure your TOTP app with this
secret.</p><a href="{{ url_for('login') }}">Login</a>''',secret=totp_secret)
    return render_template_string('''<h1>Register</h1><form
method="post"><label>Username:</label><input type="text"
name="username"><br><label>Password:</label><input type="password"
name="password"><br><label>Role:</label><select name="role"><option
value="user">User</option><option
value="admin">Admin</option></select><br><input type="submit"
value="Register"></form>''')
@app.route('/login', methods=['GET','POST'])
def login():
    if request.method=='POST':
        username=request.form.get('username')
        password=request.form.get('password')
        conn=get_db_connection()
        cur=conn.cursor()
        cur.execute('SELECT * FROM users WHERE username = ?',(username,))
        user=cur.fetchone()
        conn.close()
        if user and check_password_hash(user['password'],password):
            session['pre_2fa_user_id']=user['id']
            return redirect(url_for('two_factor'))
        else:
            return 'Invalid credentials'
    return render_template_string('''<h1>Login</h1><form
method="post"><label>Username:</label><input type="text"
name="username"><br><label>Password:</label><input type="password"
name="password"><br><input type="submit" value="Login"></form>''')
@app.route('/two_factor', methods=['GET','POST'])
def two_factor():
    if 'pre_2fa_user_id' not in session:
        return redirect(url_for('login'))
    if request.method=='POST':
        token=request.form.get('token')
        user_id=session['pre_2fa_user_id']
        conn=get_db_connection()
        cur=conn.cursor()
        cur.execute('SELECT totp_secret, role FROM users WHERE id =
?',(user_id,))
        user=cur.fetchone()
        conn.close()
        totp=pyotp.TOTP(user['totp_secret'])
        if totp.verify(token):
            session.pop('pre_2fa_user_id',None)
            session['user_id']=user_id
            session['role']=user['role']
            conn=get_db_connection()
            cur=conn.cursor()
            cur.execute('INSERT INTO access_logs (user_id, access_time, status,
details) VALUES
(?,?,?,?)',(user_id,datetime.datetime.utcnow().isoformat(),'SUCCESS','User
logged in successfully'))
            conn.commit()
            conn.close()

socketio.emit('login_notification',{'user_id':user_id,'status':'SUCCESS','time':
datetime.datetime.utcnow().isoformat()})
            return redirect(url_for('dashboard'))
        else:
            conn=get_db_connection()
```

```
            cur=conn.cursor()
            cur.execute('INSERT INTO access_logs (user_id, access_time, status,
details) VALUES
(?,?,?,?)',(user_id,datetime.datetime.utcnow().isoformat(),'FAILURE','Invalid
TOTP token'))
            conn.commit()
            conn.close()
            return 'Invalid TOTP token'
    return render_template_string('''<h1>Two-Factor Authentication</h1><form
method="post"><label>Enter TOTP Token:</label><input type="text"
name="token"><br><input type="submit" value="Verify"></form>''')
@app.route('/dashboard')
def dashboard():
    if 'user_id' not in session:
        return redirect(url_for('login'))
    conn=get_db_connection()
    cur=conn.cursor()
    cur.execute('SELECT * FROM access_logs WHERE user_id = ? ORDER BY
access_time DESC',(session['user_id'],))
    logs=cur.fetchall()
    conn.close()
    log_items=''
    for log in logs:
        log_items+='<li>ID: {} Time: {} Status: {} Details:
{}</li>'.format(log['id'],log['access_time'],log['status'],log['details'])
    admin_section=''
    if session.get('role')=='admin':
        admin_section='<p><a href="{}">View All Users
(API)</a></p>'.format(url_for('api_users'))
    return render_template_string('''<h1>Dashboard</h1><p>Welcome, user
{}</p><ul>{}</ul>{}<p><a href="{{ url_for('logout')
}}">Logout</a></p>'''.format(session['user_id'],log_items,admin_section))
@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('index'))
@app.route('/api/users')
def api_users():
    if 'user_id' not in session or session.get('role')!='admin':
        return jsonify({'error':'Unauthorized'}),401
    conn=get_db_connection()
    cur=conn.cursor()
    cur.execute('SELECT id, username, role FROM users')
    users=cur.fetchall()
    conn.close()
    users_list=[]
    for user in users:

users_list.append({'id':user['id'],'username':user['username'],'role':user['role
']})
    return jsonify(users_list)
@socketio.on('connect')
def handle_connect():
    emit('response',{'data':'Connected'})
@socketio.on('send_message')
def handle_send_message(data):
    emit('message',data,broadcast=True)
if __name__=='__main__':
    socketio.run(app,debug=True)
```

Файл certificate_management.py

```
import os
import sys
import sqlite3
import datetime
import shutil
import socket
from apscheduler.schedulers.background import BackgroundScheduler
from flask import Flask, request, send_file, redirect, url_for, session,
render_template_string, jsonify
from fpdf import FPDF
from ldap3 import Server, Connection, ALL, NTLM

DATABASE = 'extended_access_control.db'
BACKUP_FOLDER = 'db_backups'
SIEM_SERVER = '127.0.0.1'
SIEM_PORT = 514
LDAP_SERVER_URL = 'ldap://localhost'
LDAP_USER_DN = 'CN={},OU=Users,DC=example,DC=com'
LDAP_SEARCH_BASE = 'OU=Users,DC=example,DC=com'

def get_db_connection():
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn

def init_db():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("CREATE TABLE IF NOT EXISTS certificates (id INTEGER PRIMARY KEY
AUTOINCREMENT, user_id INTEGER, certificate TEXT, issue_date TEXT, valid_until
TEXT)")
    cur.execute("CREATE TABLE IF NOT EXISTS access_logs (id INTEGER PRIMARY KEY
AUTOINCREMENT, user_id INTEGER, access_time TEXT, status TEXT, details TEXT)")
    conn.commit()
    conn.close()

def automated_certificate_management():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT id, valid_until, user_id FROM certificates")
    rows = cur.fetchall()
    now = datetime.datetime.utcnow()
    for row in rows:
        valid_until = datetime.datetime.fromisoformat(row['valid_until'])
        if now >= valid_until:
            new_valid_until = now + datetime.timedelta(days=365)
            new_issue_date = now.isoformat()
            new_certificate = "AutoRenewedCert_User{}_{}".format(row['user_id'],
new_issue_date)
            cur.execute("UPDATE certificates SET certificate = ?, issue_date =
?, valid_until = ? WHERE id = ?", (new_certificate, new_issue_date,
new_valid_until.isoformat(), row['id']))
            cur.execute("INSERT INTO access_logs (user_id, access_time, status,
details) VALUES (?, ?, ?, ?)", (row['user_id'], now.isoformat(), "CERT_RENEWED",
"Certificate auto-renewed"))
    conn.commit()
    conn.close()

def generate_audit_report():
    conn = get_db_connection()
    cur = conn.cursor()
```

```
    cur.execute("SELECT id, user_id, access_time, status, details FROM
access_logs ORDER BY access_time DESC")
    logs = cur.fetchall()
    conn.close()
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", size=12)
    pdf.cell(200, 10, txt="Audit Report", ln=1, align="C")
    pdf.cell(200, 10, txt="Generated on: " +
datetime.datetime.utcnow().isoformat(), ln=2, align="C")
    pdf.ln(10)
    for log in logs:
        line = "ID: {} User: {} Time: {} Status: {} Details:
{}".format(log['id'], log['user_id'], log['access_time'], log['status'],
log['details'])
        pdf.multi_cell(0, 10, txt=line)
    report_filename =
"audit_report_{}.pdf".format(datetime.datetime.utcnow().strftime("%Y%m%d%H%M%S")
)
    pdf.output(report_filename)
    return report_filename

def backup_database():
    if not os.path.exists(BACKUP_FOLDER):
        os.makedirs(BACKUP_FOLDER)
    timestamp = datetime.datetime.utcnow().strftime("%Y%m%d%H%M%S")
    backup_file = os.path.join(BACKUP_FOLDER, "backup_{}.db".format(timestamp))
    shutil.copyfile(DATABASE, backup_file)
    return backup_file

def restore_database(backup_file):
    if os.path.exists(backup_file):
        shutil.copyfile(backup_file, DATABASE)
        return True
    return False

def ldap_authenticate(username, password):
    server = Server(LDAP_SERVER_URL, get_info=ALL)
    user_dn = LDAP_USER_DN.format(username)
    try:
        conn = Connection(server, user=user_dn, password=password,
authentication=NTLM, auto_bind=True)
        if conn.bound:
            conn.unbind()
            return True
        else:
            return False
    except Exception:
        return False

def send_logs_to_siem():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT id, user_id, access_time, status, details FROM
access_logs ORDER BY access_time DESC")
    logs = cur.fetchall()
    conn.close()
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    for log in logs:
        message = "SIEM_LOG ID:{} USER:{} TIME:{} STATUS:{}
DETAILS:{}".format(log['id'], log['user_id'], log['access_time'], log['status'],
log['details'])
        sock.sendto(message.encode('utf-8'), (SIEM_SERVER, SIEM_PORT))
```

```
    sock.close()

scheduler = BackgroundScheduler()
scheduler.add_job(automated_certificate_management, 'interval', minutes=1)
scheduler.start()

app = Flask(__name__)
app.secret_key = 'another_super_secret_key'

@app.route('/audit_report')
def audit_report():
    report_file = generate_audit_report()
    return send_file(report_file, as_attachment=True)

@app.route('/backup')
def backup():
    backup_file = backup_database()
    return jsonify({"backup_file": backup_file})

@app.route('/restore', methods=['POST'])
def restore():
    backup_file = request.form.get('backup_file')
    if restore_database(backup_file):
        return jsonify({"status": "Restore successful"})
    return jsonify({"status": "Restore failed"}), 400

@app.route('/ldap_login', methods=['GET', 'POST'])
def ldap_login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        if ldap_authenticate(username, password):
            session['ldap_user'] = username
            conn = get_db_connection()
            cur = conn.cursor()
            cur.execute("INSERT INTO access_logs (user_id, access_time, status,
details) VALUES (?, ?, ?, ?)", (0, datetime.datetime.utcnow().isoformat(),
"LDAP_SUCCESS", "LDAP login for user " + username))
            conn.commit()
            conn.close()
            return redirect(url_for('dashboard_ldap'))
        else:
            conn = get_db_connection()
            cur = conn.cursor()
            cur.execute("INSERT INTO access_logs (user_id, access_time, status,
details) VALUES (?, ?, ?, ?)", (0, datetime.datetime.utcnow().isoformat(),
"LDAP_FAILURE", "LDAP login failed for user " + username))
            conn.commit()
            conn.close()
            return 'LDAP Authentication Failed'
    return render_template_string('<form method="post"><input type="text"
name="username" placeholder="Username"><br><input type="password"
name="password" placeholder="Password"><br><input type="submit" value="LDAP
Login"></form>')

@app.route('/dashboard_ldap')
def dashboard_ldap():
    if 'ldap_user' not in session:
        return redirect(url_for('ldap_login'))
    return render_template_string('<h1>Welcome LDAP User {{ user }}</h1><p><a
href="{{ url_for(\'logout_ldap\') }}">Logout</a></p>',
user=session['ldap_user'])
```

```
@app.route('/logout_ldap')
def logout_ldap():
    session.pop('ldap_user', None)
    return redirect(url_for('ldap_login'))

@app.route('/siem_send')
def siem_send():
    send_logs_to_siem()
    return jsonify({"status": "Logs sent to SIEM"})

@app.route('/')
def index():
    return render_template_string('<h1>Extended System</h1><ul><li><a href="{{
url_for(\'audit_report\') }}">Download Audit Report</a></li><li><a href="{{
url_for(\'backup\') }}">Backup Database</a></li><li><a href="{{
url_for(\'ldap_login\') }}">LDAP Login</a></li><li><a href="{{
url_for(\'siem_send\') }}">Send Logs to SIEM</a></li></ul>')

if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

Файл encryption_key.py

```
import os
import sys
import sqlite3
import datetime
import threading
import time
import random
import hashlib
from flask import Flask, request, session, render_template_string, redirect,
url_for, jsonify
from cryptography.fernet import Fernet
import cv2
import numpy as np
app = Flask(__name__)
app.secret_key = 'super_secret_key_for_mobile_and_biometric'
DATABASE = 'extended_features.db'
def get_db_connection():
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn
def init_db():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("CREATE TABLE IF NOT EXISTS biometric_users (id INTEGER PRIMARY
KEY AUTOINCREMENT, username TEXT UNIQUE, biometric_hash TEXT)")
    cur.execute("CREATE TABLE IF NOT EXISTS encrypted_notes (id INTEGER PRIMARY
KEY AUTOINCREMENT, user_id INTEGER, note_encrypted TEXT, created_at TEXT)")
    cur.execute("CREATE TABLE IF NOT EXISTS network_status (id INTEGER PRIMARY
KEY AUTOINCREMENT, ip TEXT, status TEXT, response_time REAL, checked_at TEXT)")
    conn.commit()
    conn.close()
init_db()
ENCRYPTION_KEY_FILE = 'fernet.key'
def load_encryption_key():
    if os.path.exists(ENCRYPTION_KEY_FILE):
        with open(ENCRYPTION_KEY_FILE, 'rb') as f:
            key = f.read()
    else:
        key = Fernet.generate_key()
        with open(ENCRYPTION_KEY_FILE, 'wb') as f:
            f.write(key)
    return key
fernet_key = load_encryption_key()
fernet = Fernet(fernet_key)
@app.route('/mobile_login', methods=['GET','POST'])
def mobile_login():
    if request.method=='POST':
        username = request.form.get('username')
        password = request.form.get('password')
        if username=='mobileuser' and password=='mobilepass':
            session['mobile_user'] = username
            return redirect(url_for('mobile_dashboard'))
        else:
            return render_template_string("<h1>Mobile Login Failed</h1><a
href='/mobile_login'>Try Again</a>")
    return render_template_string("<h1>Mobile Login</h1><form
method='post'><input type='text' name='username'
placeholder='Username'><br><input type='password' name='password'
placeholder='Password'><br><input type='submit' value='Login'></form>")
@app.route('/mobile_dashboard')
def mobile_dashboard():
```

```
    if 'mobile_user' not in session:
        return redirect(url_for('mobile_login'))
    return render_template_string("<h1>Mobile Dashboard</h1><p>Welcome {{ user
}}</p><a href='/mobile_logout'>Logout</a>", user=session['mobile_user'])
@app.route('/mobile_logout')
def mobile_logout():
    session.pop('mobile_user', None)
    return redirect(url_for('mobile_login'))
@app.route('/biometric_register', methods=['GET','POST'])
def biometric_register():
    if request.method=='POST':
        username = request.form.get('username')
        file = request.files.get('biometric')
        if file:
            file_bytes = file.read()
            image_array = np.frombuffer(file_bytes, np.uint8)
            img = cv2.imdecode(image_array, cv2.IMREAD_GRAYSCALE)
            if img is None:
                return render_template_string("<h1>Invalid image</h1><a
href='/biometric_register'>Try Again</a>")
            resized = cv2.resize(img, (100,100))
            hash_val = hashlib.sha256(resized.tobytes()).hexdigest()
            conn = get_db_connection()
            cur = conn.cursor()
            try:
                cur.execute("INSERT INTO biometric_users (username,
biometric_hash) VALUES (?, ?)", (username, hash_val))
                conn.commit()
            except sqlite3.IntegrityError:
                conn.close()
                return render_template_string("<h1>User already
registered</h1><a href='/biometric_register'>Try Again</a>")
            conn.close()
            return render_template_string("<h1>Biometric Registration
Successful</h1><a href='/biometric_login'>Login</a>")
        else:
            return render_template_string("<h1>No file uploaded</h1><a
href='/biometric_register'>Try Again</a>")
    return render_template_string("<h1>Biometric Registration</h1><form
method='post' enctype='multipart/form-data'><input type='text' name='username'
placeholder='Username'><br><input type='file' name='biometric'><br><input
type='submit' value='Register'></form>")
@app.route('/biometric_login', methods=['GET','POST'])
def biometric_login():
    if request.method=='POST':
        username = request.form.get('username')
        file = request.files.get('biometric')
        if file:
            file_bytes = file.read()
            image_array = np.frombuffer(file_bytes, np.uint8)
            img = cv2.imdecode(image_array, cv2.IMREAD_GRAYSCALE)
            if img is None:
                return render_template_string("<h1>Invalid image</h1><a
href='/biometric_login'>Try Again</a>")
            resized = cv2.resize(img, (100,100))
            hash_val = hashlib.sha256(resized.tobytes()).hexdigest()
            conn = get_db_connection()
            cur = conn.cursor()
            cur.execute("SELECT biometric_hash FROM biometric_users WHERE
username = ?", (username,))
            row = cur.fetchone()
            conn.close()
            if row and row['biometric_hash'] == hash_val:
```

```
                session['biometric_user'] = username
                return redirect(url_for('biometric_dashboard'))
            else:
                return render_template_string("<h1>Biometric Authentication
Failed</h1><a href='/biometric_login'>Try Again</a>")
        else:
            return render_template_string("<h1>No file uploaded</h1><a
href='/biometric_login'>Try Again</a>")
    return render_template_string("<h1>Biometric Login</h1><form method='post'
enctype='multipart/form-data'><input type='text' name='username'
placeholder='Username'><br><input type='file' name='biometric'><br><input
type='submit' value='Login'></form>")
@app.route('/biometric_dashboard')
def biometric_dashboard():
    if 'biometric_user' not in session:
        return redirect(url_for('biometric_login'))
    return render_template_string("<h1>Biometric Dashboard</h1><p>Welcome {{
user }}</p><a href='/biometric_logout'>Logout</a>",
user=session['biometric_user'])
@app.route('/biometric_logout')
def biometric_logout():
    session.pop('biometric_user', None)
    return redirect(url_for('biometric_login'))
@app.route('/add_encrypted_note', methods=['GET','POST'])
def add_encrypted_note():
    if request.method=='POST':
        user_id = request.form.get('user_id')
        note = request.form.get('note')
        encrypted_note = fernet.encrypt(note.encode()).decode()
        created_at = datetime.datetime.utcnow().isoformat()
        conn = get_db_connection()
        cur = conn.cursor()
        cur.execute("INSERT INTO encrypted_notes (user_id, note_encrypted,
created_at) VALUES (?, ?, ?)", (user_id, encrypted_note, created_at))
        conn.commit()
        conn.close()
        return redirect(url_for('get_encrypted_notes'))
    return render_template_string("<h1>Add Encrypted Note</h1><form
method='post'><input type='text' name='user_id' placeholder='User
ID'><br><textarea name='note' placeholder='Note'></textarea><br><input
type='submit' value='Add'></form>")
@app.route('/get_encrypted_notes')
def get_encrypted_notes():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT id, user_id, note_encrypted, created_at FROM
encrypted_notes")
    rows = cur.fetchall()
    conn.close()
    notes = []
    for row in rows:
        decrypted = fernet.decrypt(row['note_encrypted'].encode()).decode()
        notes.append({"id": row['id'], "user_id": row['user_id'], "note":
decrypted, "created_at": row['created_at']})
    return jsonify(notes)
def monitor_network():
    ips = ["192.168.1.1", "192.168.1.2", "8.8.8.8", "8.8.4.4"]
    while True:
        conn = get_db_connection()
        cur = conn.cursor()
        for ip in ips:
            response_time = random.uniform(10, 200)
            status = "UP" if random.choice([True, False]) else "DOWN"
```

```
            checked_at = datetime.datetime.utcnow().isoformat()
            cur.execute("INSERT INTO network_status (ip, status, response_time,
checked_at) VALUES (?, ?, ?, ?)", (ip, status, response_time, checked_at))
        conn.commit()
        conn.close()
        time.sleep(30)
network_thread = threading.Thread(target=monitor_network, daemon=True)
network_thread.start()
@app.route('/network_status')
def network_status():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute("SELECT id, ip, status, response_time, checked_at FROM
network_status ORDER BY checked_at DESC LIMIT 20")
    rows = cur.fetchall()
    conn.close()
    status_list = []
    for row in rows:
        status_list.append({"id": row['id'], "ip": row['ip'], "status":
row['status'], "response_time": row['response_time'], "checked_at":
row['checked_at']})
    return jsonify(status_list)
if __name__=='__main__':
    app.run(debug=True)
```