Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«МЕТОД ТА ЗАСІБ ЗАХИСТУ ПРОГРАМНОГО ЗАСТОСУНКУ З АПАРАТНОЮ ПРИВ'ЯЗКОЮ ДО USB-НОСІЇВ»

Виконав: студент 2 курсу групи 1БС-24 м
спеціальності 125 Кібербезпека та захист інформації
_____ Олексій ЛЕСЬКО

Керівник: к. т. н., доц., доцент каф. ЗІ
_____ Віталій ЛУКІЧОВ
« 19 » грудня 2025 р.

Опонент: к. т. н., доцент каф. ТВ
_____ Володимир МАЙДАНЮК
« 19 » грудня 2025 р.

Допущено до захисту
В. о. зав. каф. ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
« 19 » грудня 2025 р.

Вінниця ВНТУ – 2025 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека та захист інформації
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
В. о. зав. каф. ЗІ,
д. т. н., проф.
_____Володимир ЛУЖЕЦЬКИЙ
«24» 09 2025 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Лесько Олексію Віталійовичу

1. Тема роботи: «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв»
   керівник роботи: Віталій ЛУКІЧОВ, к. т. н., доцент кафедри ЗІ, затверджений наказом ректора ВНТУ від 24 вересня 2025 року № 313.
2. Строк подання студентом роботи19 грудня 2025 р.
3. Вихідні дані до роботи:
   – Мова програмування Python.
   – Об'єкт розробки – програмний модуль захисту ПЗ з функціями криптографії, автентифікації, Trust Score та Recovery.
4. Зміст текстової частини: Вступ. 1. Аналіз інформаційних джерел. 2. Розробка методу захисту програмного застосунку з прив'язкою до USB-носіїв. 3. Розробка програмного засобу захисту з апаратною прив'язкою до USB -носіїв. 4. Економічна частина. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Архітектура системи захисту програмного застосунку. Алгоритм генерації та розподілу часток секрету. Алгоритм відновлення секрету за схемою шаміра. Алгоритм адаптивного визначення порогу k. Діаграма станів системи адаптивного визначення порогу k. Алгоритм автентифікації користувача. Результати тестування інтерфейсу. Результати модульного тестування. Показники економічної ефективності.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Віталій ЛУКІЧОВ, к.т.н., доц. каф. ЗІ	25.09.2025	18.12.25
2	Віталій ЛУКІЧОВ, к.т.н., доц. каф. ЗІ	25.09.2025	18.12.25
3	Віталій ЛУКІЧОВ, к.т.н., доц. каф. ЗІ	25.09.2025	18.12.25
5	Олександр ЛЕСЬКО, зав. каф. ЕПВМ, к. е. н., доц.	18.11.2025	18.12.25

7. Дата видачі завдання 24 вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітки
1	Аналіз завдання. Вступ	24.09.2025 – 26.09.2025	
2	Аналіз інформаційних джерел за напрямком магістерської кваліфікаційної роботи	27.09.2025 – 07.09.2025	
3	Науково-технічне обґрунтування	11.10.2025 – 22.10.2025	
4	Розробка технічного завдання	23.10.2025 – 26.10.2025	
5	Аналіз методів та засобів захисту програмного застосунку	27.10.2025 – 02.11.2025	
6	Аналіз криптографічних алгоритмів AES, SHA-256 та схеми поділу секрету Шаміра	03.11.2025 – 07.11.2025	
7	Розробка моделі оцінки довіри (Trust Score) та алгоритму адаптивного визначення порогу k	08.11.2025 – 14.11.2025	
8	Проєктування архітектури системи захисту з апаратною прив'язкою до USB-носіїв	15.11.2025 – 17.11.2025	
9	Розробка розділу економічного обґрунтування доцільності розробки	18.11.2025 – 22.11.2025	
10	Аналіз виконання ТЗ, висновки	23.11.2025 – 28.11.2025	
11	Оформлення пояснювальної записки	29.11.2025 – 11.12.2025	
12	Перевірка магістерської роботи на наявність текстових запозичень	12.12.2025 – 15.12.2025	
13	Попередній захист та доопрацювання МКР	03.12.2025 – 14.12.2025	
14	Представлення МКР до захисту, рецензування	17.12.2025 – 18.12.2025	
15	Захист МКР	19.12.2025 – 23.12.2025	

Студент _____ Олексій ЛЕСЬКО

Керівник роботи _____ Віталій ЛУКІЧОВ

АНОТАЦІЯ

УДК 004.056

Лесько О. Метод та засіб захисту програмного застосунку з апаратною прив’язкою до USB-носіїв. Магістерська кваліфікаційна робота зі спеціальності 125 – Кібербезпека та захист інформації, освітня програма – Безпека інформаційних і комунікаційних систем. Вінниця: ВНТУ, 2025. 88 с.

Укр. мовою. Бібліогр.: 28 назв; рис. 22; табл.: 21.

Магістерська кваліфікаційна робота присвячена розробці методу та програмного засобу захисту програмного застосунку з апаратною прив’язкою до USB-носіїв. Проаналізовано сучасні загрози та обмеження традиційних статичних методів апаратної і багатофакторної автентифікації.

Запропоновано адаптивний метод захисту на основі порогової схеми Шаміра з динамічним визначенням кількості факторів автентифікації залежно від рівня довіри (Trust Score). Метод використовує до п’яти незалежних факторів, що унеможливлює компрометацію секрету при втраті окремих з них.

Розроблено архітектуру системи та програмний засіб реалізації мовою Python з використанням бази даних SQLite. Тестування підтвердило суттєве зростання стійкості до несанкціонованого доступу порівняно з класичними методами апаратної прив’язки.

Ілюстративна частина складається з 9 плакатів з демонстрацією результатів проведеного тестування.

Ключові слова: апаратна прив’язка, USB-носій, багатофакторна автентифікація, схема Шаміра, Trust Score, відновлення доступу, кібербезпека, захист програмного застосунку.

ABSTRACT

UDC 004.056

Lesko O. Method and software tool for protecting software with hardware binding to USB storage devices**.** Master's qualification work in specialty 125 – Cybersecurity and Information Protection, educational program Information and Communication Systems Security. Vinnytsia: VNTU, 2025. 88 p.

In Ukrainian language. Bibliographer: 28 titles; 22 figures; 21 tables.

The master's qualification work is devoted to the development of a method and a software tool for protecting software using hardware binding to USB storage devices. Modern threats to software security and the limitations of traditional static approaches to hardware-based and multi-factor authentication were analyzed.

An adaptive protection method based on Shamir's threshold secret sharing scheme with dynamic determination of the required number of authentication factors depending on the trust level (Trust Score) is proposed. The method employs up to five independent authentication factors, which prevents secret compromise in the event of individual factor loss.

The system architecture and a software implementation developed in Python using an SQLite database are presented. Testing results confirmed a significant increase in resistance to unauthorized access compared to classical hardware binding methods.

The illustrative part consists of 9 posters demonstrating the testing results.

Keywords: hardware binding, USB storage device, multi-factor authentication, Shamir's scheme, Trust Score, access recovery, cybersecurity, software protection.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі USB-носії є одним із найпоширеніших засобів зберігання та передачі даних, забезпечуючи мобільність і зручність користувачів. Проте водночас вони залишаються потенційним джерелом кіберзагроз, зокрема поширення шкідливого програмного забезпечення та несанкціонованого доступу до інформації. За даними Honeywell (2024), понад 51% кібератак із використанням шкідливого ПЗ здійснюється саме через USB-пристрої, що свідчить про необхідність удосконалення систем захисту.

Сучасні методи безпеки USB-пристроїв зазвичай зосереджуються або на апаратному, або на програмному рівні. Однак лише поєднання обох підходів дає можливість створити справді стійкі рішення, які враховують як автентифікацію пристрою, так і перевірку особистості користувача. Такі рішення можуть бути особливо ефективними у сфері захисту ліцензованого програмного забезпечення, де важливо гарантувати, що ПЗ працює лише на авторизованих пристроях.

Дослідження науковців D.V. Pham, K. Lee, O.A. Ibrahim, F. Griscioli та M. Pizzonia [1-5] показали, що комбіновані підходи на основі апаратної автентифікації, криптографічного захисту та поведінкових характеристик значно підвищують безпеку інформаційних систем. Ці напрацювання створили підґрунтя для подальших розробок у напрямі апаратно-програмного захисту, що й визначає актуальність даного дослідження.

Метою магістерської роботи є підвищення ефективності захисту програмного забезпечення від несанкціонованого використання шляхом розробки методу апаратної прив’язки до USB-носіїв із використанням адаптивної схеми поділу секрету Шаміра та динамічної оцінки довіри до факторів автентифікації.

Для досягнення мети необхідно виконати такі **завдання**:

- проаналізувати існуючі методи та засоби захисту програмного забезпечення з прив’язкою до USB-носіїв;
- узагальнено представити метод захисту програмного забезпечення;
- розробити архітектуру засобу;
- розробити алгоритми, що реалізують метод;
- реалізувати засіб захисту програмного забезпечення на основі методу;
- протестувати коректність та безпеку програмного засобу;
- визначити показники економічної ефективності розробки.

Об’єктом дослідження є процеси зберігання та передачі даних за допомогою USB-носіїв, а також механізми взаємодії користувача з апаратно-програмними засобами захисту.

Предметом дослідження є методи і засоби підвищення рівня безпеки програмного забезпечення шляхом використання апаратної прив’язки до USB-носіїв та криптографічного захисту на основі порогових схем поділу секрету.

Наукова новизна роботи полягає у створенні комплексного підходу до захисту програмного забезпечення, який передбачає:

- впровадження адаптивної схеми поділу секрету Шаміра з динамічно змінним порогом доступу;
- розробку механізму оцінки довіри (Trust Score) для факторів автентифікації;
- поєднання USB-відбитка, Hardware ID, пароля користувача та TOTP у єдину криптографічну систему;
- реалізацію безпечного механізму відновлення доступу без втрати цілісності системи безпеки.

Практичне значення полягає у створенні багаторівневого засобу захисту програмного забезпечення, який забезпечує контроль доступу, запобігає несанкціонованому копіюванню, виявляє аномальні спроби входу в реальному часі та реалізує безпечний механізм відновлення доступу.

1 АНАЛІЗ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1.1 Аналіз сучасних загроз програмному забезпеченню

У сучасному цифровому середовищі програмне забезпечення стало одним із найцінніших активів як для окремих користувачів, так і для комерційних та державних структур. Воно забезпечує автоматизацію процесів, управління даними, обмін інформацією та реалізацію ключових функцій бізнесу. У зв’язку з цим програмне забезпечення часто стає ціллю атак з боку зловмисників, які прагнуть отримати несанкціонований доступ до його функцій, скопіювати або модифікувати його з комерційною або шкідливою метою.

Проте стрімкий розвиток інформаційних технологій, глобалізація мережевого доступу призвели до збільшення кількості й складності загроз, що впливають на безпеку програмного забезпечення. Особливу небезпеку становлять високотехнологічні атаки, зокрема з використанням реверс-інжинірингу, шкідливого коду, віртуального середовища для аналізу, тощо.

До основних загроз можна віднести [1]:

- несанкціоноване копіювання;
- реверс-інжиніринг;
- модифікація коду;
- запуск на непередбачених пристроях;
- ін’єкція шкідливого коду або дебагінг;
- використання віртуального середовища для обходу захисту;
- аналіз з використанням емуляторів або sandbox-середовищ.

Основні загрози наведено у табл. 1.1 [1-2].

З кожною з цих загроз пов’язано певні негативні наслідки: втрата доходів розробника, порушення авторських прав, витік логіки та конфіденційних даних, підміна функціональності, створення нелегальних копій або аналогів тощо.

Таблиця 1.1 – Основні загрози та методи захисту програмного забезпечення

№	Тип загрози	Приклад	Наслідки	Методи протидії
1	Піратство	Розповсюдження exe-файлу без ліцензії	Втрата прибутку, порушення прав	DRM, серійні ключі, онлайн-активація
2	Реверс-інжиніринг	Аналіз через Ghidra	Витік логіки, створення клонів	Обфускація, упаковка, перевірка дебагера
3	Патчинг	Видалення ліцензійної перевірки	Несанкціонований доступ	Контроль цілісності (SHA256), цифровий підпис
4	Несанкціонований запуск	Запуск на іншому ПК	Порушення ліцензії, поширення програмного забезпечення	Прив’язка до флеш-носія або MAC-адреси
5	Код-ін’єкція	DLL injection	Перехоплення функцій, компрометація	Антидебаг, захист від ін’єкцій
6	Запуск у VM	Sandbox-аналіз	Злом та вивчення поведінки	Виявлення VM, заборона запуску у віртуальному середовищі

Крім зазначених загроз, сучасне програмне забезпечення піддається впливу складніших атак, таких як цільові кібер-атки, соціальна інженерія та експлуатація нульових вразливостей (zero-day). Вони дозволяють зловмисникам отримати доступ до критично важливої інформації без попередження та обходити традиційні методи захисту. Зростає роль автоматизованих інструментів для аналізу поведінки програмного забезпечення, які здатні виявляти аномалії, несанкціоновані модифікації або спроби обходу захисту в реальному часі.

Сучасні підходи до протидії загрозам поєднують апаратні та програмні методи, включаючи криптографічний захист, системи контролю цілісності, обфускацію коду, ліцензійні перевірки та апаратну прив’язку до конкретних пристроїв. Використання таких комплексних рішень дозволяє мінімізувати ризики втрати даних, витоку комерційної інформації та порушення авторських прав, забезпечуючи високий рівень безпеки для кінцевих користувачів і організацій.

Надалі, для ефективного управління загрозами, важливо впроваджувати системи моніторингу та аудиту, які дозволяють виявляти підозрілі дії та оперативно реагувати на потенційні атаки, що значно підвищує стійкість програмного забезпечення до сучасних кіберзагроз.

1.2 Огляд існуючих рішень для захисту програмного забезпечення з апаратною прив'язкою

Одним із найбільш розповсюджених методів боротьби з несанкціонованим використанням програмного забезпечення є застосування апаратної прив’язки до певного фізичного носія. Такий підхід дозволяє обмежити запуск програмного продукту лише на комп’ютерах, які мають доступ до захищеного пристрою (USB-носія, апаратного токена або спеціалізованого криптопроцесора).

Серед відомих рішень у сфері захисту ПЗ з апаратною прив’язкою можна виділити кілька основних груп:

1. USB-донгли (апаратні ключі).

Найбільш поширеним рішенням є використання USB-донглів, які виступають у ролі апаратного ідентифікатора, як наведено на рисунку 1.1. Донгл містить у собі унікальний серійний номер та, у деяких випадках, криптографічний модуль. Програмне забезпечення перевіряє наявність цього пристрою при запуску, і лише у випадку успішної перевірки відбувається доступ до функціоналу.

Приклади комерційних рішень:

– Hardware Against Software Piracy (HASP) – одна з найстаріших та найбільш відомих систем. HASP-донгли підтримують апаратне шифрування та автентифікацію, що дозволяє реалізовувати ліцензування на рівні функціоналу програмного продукту [3].

– Guardant – система захисту ПЗ, що поєднує апаратні ключі та сервери ліцензування. Використовує криптографію на рівні ядра та інтегрується з .NET, Java, C++ додатками [4].

– Eutron SmartKey – апаратні ключі з підтримкою алгоритмів AES та SHA-1, що забезпечують надійність при перевірці ліцензії [5].

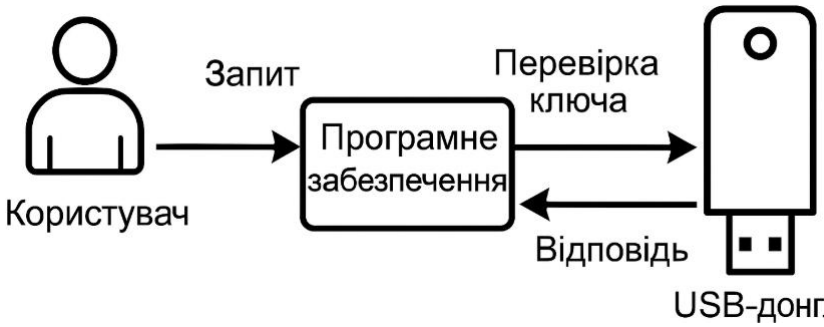


Рисунок 1.1 – Схема роботи програмного забезпечення з USB-донглом

2. Апаратна прив’язка до серійних номерів носіїв.

Більш простим методом є перевірка унікального серійного номера флеш-накопичувача або жорсткого диска, як щображено на рисунку 1.2. У цьому випадку система не потребує спеціалізованого апаратного забезпечення, а лише зчитує ідентифікаційні дані пристрою.

Недоліком такого підходу є відносно низький рівень захисту: серійний номер може бути змінений програмними засобами або підроблений. Проте цей метод є дешевшим у впровадженні й широко використовується у внутрішніх корпоративних рішеннях [6].

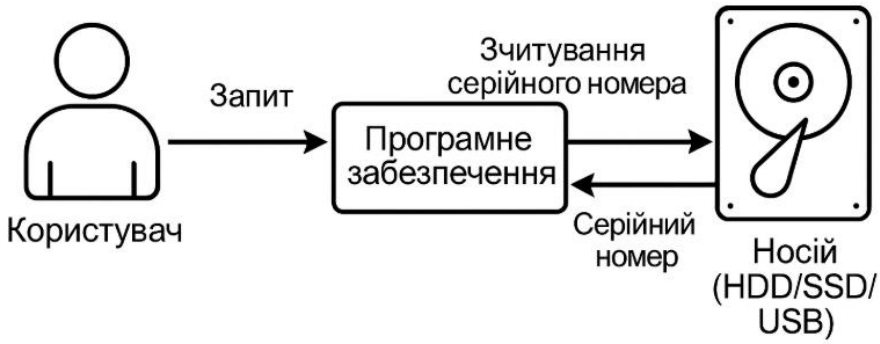


Рисунок 1.2 – Схема перевірки серійного номера носія

3. Використання смарт-карт та криптотокенів

У більш захищених системах застосовуються смарт-карти або криптотокени, що наведено на рисунку 1.3. Такі пристрої мають вбудовані чипи з підтримкою апаратних криптографічних операцій (RSA, AES, ECC). Вони дозволяють не лише перевіряти наявність ключа, а й зберігати криптографічні секрети всередині пристрою без можливості їх експорту.

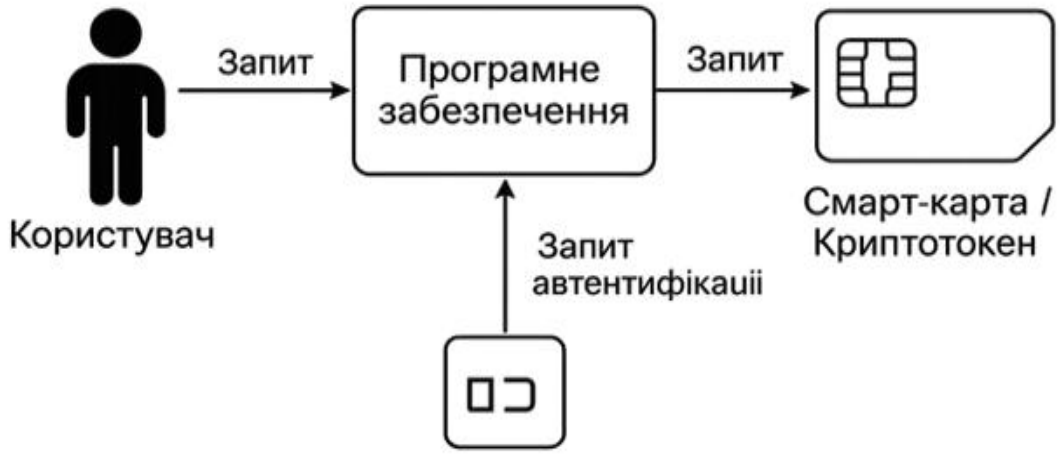


Рисунок 1.3 – Схема використання смарт-карти або криптотокена для захисту ПЗ

Приклади смарт-карт та криптотокенів:

– SafeNet eToken – криптотокени, які забезпечують двофакторну автентифікацію та зберігання приватних ключів. Використовуються як у корпоративних рішеннях, так і в державних системах [7].

– YubiKey – універсальні апаратні ключі безпеки, що підтримують протоколи FIDO2, U2F, OTP та смарт-картовий режим. Застосовуються для захисту облікових записів, VPN-доступу та у системах з багатофакторною автентифікацією.

– Gemalto IDPrime – смарт-карти з інтегрованою підтримкою PKI-інфраструктури, використовуються у банківській сфері та для електронного цифрового підпису.

4. Комбіновані системи (апаратна прив’язка + мережеві сервери ліцензій)

Окрему групу складають рішення, які поєднують апаратну перевірку з авторизацією через мережевий сервер, що зображено на рисунку 1.4. Такий підхід дозволяє централізовано керувати ліцензіями, але вимагає наявності підключення до Інтернету або внутрішньої корпоративної мережі.

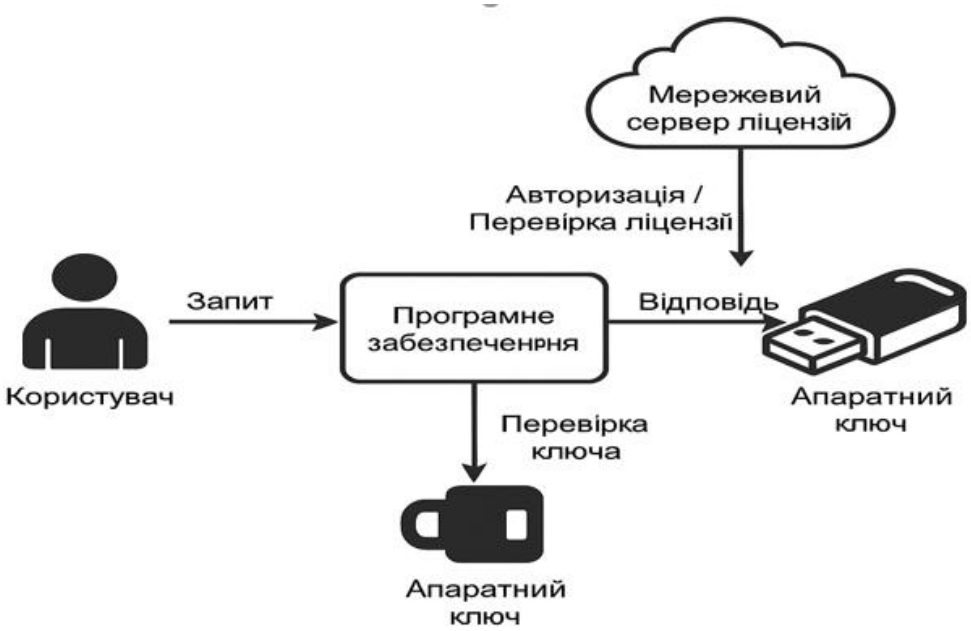


Рисунок 1.4 – Схема комбінованої системи захисту з апаратною прив’язкою та мережевим сервером ліцензій

Для прикладу, Sentinel LDK – система, яка об’єднує локальний апаратний ключ і хмарний сервер ліцензування [8].

Таким чином, огляд показує, що найбільш захищеними є рішення, засновані на апаратних токенах з вбудованими криптомодулями, тоді як

прив’язка до серійних номерів є лише базовим рівнем захисту. Вибір конкретної технології залежить від вартості впровадження, рівня вимог до безпеки та цільової аудиторії програмного продукту.

1.3 Методи криптографічного захисту програмного забезпечення

Для боротьби з загрозами застосовуються спеціалізовані методи, що охоплюють як технічні, так і організаційні аспекти. Серед них основну роль відіграють криптографічні алгоритми (наприклад, AES, SHA-256), які забезпечують шифрування даних і контроль цілісності. Також важливими є методи апаратної прив’язки до конкретного пристрою, як-от флеш-носія, що дозволяє запобігти запуску програмного забезпечення на неавторизованих системах. Додатковими способами є обфускація коду, цифрові підписи, ліцензійна перевірка та системи виявлення змін (Tamper Detection) [9].

Методи захисту програмного забезпечення формують комплекс заходів, спрямованих на запобігання несанкціонованому копіюванню, зміненню, запуску або зворотному інжинірингу програмного продукту. Сучасна кібербезпека активно розвиває цей напрямок, оскільки уразливості в програмному забезпеченні можуть призводити до серйозних витоків даних, порушення ліцензійних угод та фінансових збитків.

Основні категорії методів захисту:

1. Криптографічні методи – використання криптографічних алгоритмів ключовий елемент захисту.

– AES (Advanced Encryption Standard): симетричний алгоритм, визнаний стандартом шифрування в державних та комерційних системах [9].

– SHA-256: геш-функція, яка не виконує шифрування, але дозволяє перевіряти цілісність та аутентифікацію даних, зокрема файлів або ключових даних [5].

– Схема поділу секрету Шаміра (Shamir’s Secret Sharing, SSS) – метод розподілу секретного значення (наприклад, ключа шифрування або ліцензійного

коду) між кількома учасниками таким чином, що для його відновлення достатньо лише певної кількості частин (*k із n*). Такий підхід виключає необхідність зберігання ключа у відкритому вигляді, підвищуючи стійкість системи до компрометації. У класичній схемі поріг *k* є статичним і визначається під час створення частин, що обмежує гнучкість при зміні умов доступу. Попри це, схема Шаміра є ефективним і широко застосовуваним криптографічним методом у системах управління ключами та захисту програмного забезпечення [25].

2. Апаратна прив’язка (Hardware Binding) – прив’язка роботи програмного забезпечення до апаратного носія, полягає в перевірці унікального ідентифікатора пристрою (серійного номера або Volume ID). Це дозволяє запобігти запуску програми на неавторизованих комп’ютерах. Такий метод складно обійти без фізичного доступу до носія, тому він широко використовується в розробці платного програмного забезпечення, особливо офлайн-систем [10].

3. Обфускація коду (Code Obfuscation) – цей метод перетворює вихідний код у вигляд, важкий для аналізу або модифікації, без зміни логіки його роботи. Обфускація може бути реалізована на рівні байт-коду Java за допомогою таких інструментів, як ProGuard або Allatori.

4. Ліцензійні ключі та активація – один із найрозповсюдженіших методів у комерційних продуктах – перевірка ліцензійного ключа або коду активації, збереженого у зашифрованому вигляді. Під час запуску ключ перевіряється та порівнюється з параметрами системи [11].

5. Цифрові підписи та сертифікати – підписані файли гарантують, що вони не були змінені після створення. Це особливо важливо для оновлень та системних компонентів. Застосовується здебільшого у великих проєктах.

6. Системи виявлення модифікації (Tamper Detection) – всі важливі компоненти програмного забезпечення перевіряються на відповідність еталонним геш-значенням. У разі виявлення змін виконання припиняється. У твоєму проєкті цю роль виконує SHA-256.

7. Захист через обмеження середовища запуску – програма перевіряє, чи не працює вона у віртуальній машині, зневаднику або середовищі аналізу, і припиняє роботу, якщо виявлено спробу втручання. Цей підхід не реалізовано у поточному рішенні, але є перспективним для розширення функціональності.

У рамках магістерської роботи реалізовано комбінацію найбільш ефективних на поточному етапі методів захисту програмного забезпечення – криптографічне шифрування AES, гешування SHA-256 та апаратна прив'язка до флеш-носія. Це дозволяє забезпечити як конфіденційність, так і контроль доступу до програмного продукту.

У сучасних умовах захисту інформації важливо не лише забезпечити шифрування даних, а й контролювати доступ до зашифрованого ресурсу. Одним із ефективних методів є прив'язка доступу до конкретного фізичного носія – наприклад, USB-носія. Такий підхід забезпечує двофакторну автентифікацію: поєднання програмного компонента (наприклад, логін/пароль або ключ шифрування) із апаратним (фізичним) ідентифікатором.

Причини вибору USB-носія:

1. Мобільність і поширеність – USB-накопичувачі є доступними, компактними, не потребують спеціального обладнання та сумісні з більшістю сучасних комп'ютерів.

2. Унікальність серійного номера – кожен флеш-носій має унікальний серійний номер або ідентифікатор, який може бути використаний як частина механізму аутентифікації. Це дозволяє точно ідентифікувати носій та виключити підробки.

3. Неможливість простого копіювання – на відміну від файлів, серійний номер USB-пристрою не можна просто скопіювати або змінити без спеціалізованого обладнання.

4. Відсутність постійного підключення – прив'язка до USB дозволяє легко керувати доступом до зашифрованого ресурсу: без підключення носія система не зможе розшифрувати файли, навіть за наявності програми. Всі можливі методи прив'язки до флеш-носія наведено в табл.1.2 [12-14].

Таблиця 1.2 – Методи прив’язки до USB-носія

Метод	Опис	Переваги	Недоліки
За серійним номером	Порівняння серійного номера носія із записаним значенням	Простота, швидкість	Може бути змінено спеціальними утилітами
За хешем вмісту сектора	Хешується визначена частина вмісту флеш-носія	Складніше підробити	Залежність від структури носія
Комбінований (серійний + хеш)	Поєднання серійного номера і контрольного хешу	Підвищена надійність	Складніша реалізація

1.4 Апаратна прив'язка програмного забезпечення переваги та недоліки

Апаратна прив’язка програмного забезпечення є одним із найбільш поширених і ефективних методів захисту від несанкціонованого копіювання та використання програмних продуктів. Сутність цього підходу полягає у тому, що доступ до функціоналу системи можливий лише за умови наявності певного апаратного ідентифікатора. Такими ідентифікаторами можуть виступати USB-донгли, смарт-карти, криптотокени або унікальні серійні номери носіїв. Це поєднання апаратних та програмних механізмів значно ускладнює несанкціоноване відтворення або модифікацію програмного продукту, підвищуючи загальний рівень безпеки.

1.4.1 Переваги апаратної прив’язки.

Першою суттєвою перевагою є високий рівень захисту від піратства. Програмне забезпечення, захищене апаратним ключем, не може бути запущене без відповідного пристрою, навіть якщо копія коду була незаконно скопійована або передана. Це забезпечує надійний контроль доступу та знижує ризик незаконного поширення програмного продукту. У корпоративних та державних

системах це дозволяє ефективно регулювати кількість одночасних користувачів та забезпечувати дотримання ліцензійних умов.

Другий аспект переваг полягає у можливості реалізації криптографічних алгоритмів всередині апаратного носія. Використання алгоритмів AES, RSA або ECC дозволяє зберігати криптографічні ключі та інші секрети без можливості їх експорту з пристрою, що гарантує автентичність та конфіденційність ліцензійних даних. Таке поєднання апаратного та криптографічного захисту забезпечує подвійний рівень безпеки і знижує ймовірність несанкціонованого доступу до програмного забезпечення.

Третьою перевагою є підтримка різних моделей ліцензування. Апаратна прив’язка дозволяє реалізовувати активацію окремих функцій програмного продукту, обмежувати кількість робочих місць, надавати тимчасові або пробні ліцензії. Це робить систему гнучкою і зручною для корпоративного середовища, де часто потрібна централізована система контролю доступу та інтеграція з PKI-інфраструктурою.

Четверта перевага полягає у забезпеченні зручності використання для організацій із високими вимогами до безпеки. Централізоване управління апаратними ключами, багаторівнева автентифікація та підтримка різних рівнів доступу роблять систему ефективною для великих корпоративних, державних та фінансових проєктів.

Окрім того, апаратна прив’язка дозволяє реалізовувати додаткові механізми контролю безпеки. Наприклад, можна відстежувати частоту використання USB-ключів, реєструвати спроби запуску програми на неавторизованих пристроях або автоматично блокувати доступ при виявленні підозрілих дій. Ці функції значно підвищують захищеність системи і дозволяють оперативно реагувати на потенційні загрози.

1.4.2 Недоліки апаратної прив’язки.

Серед основних недоліків варто відзначити високу вартість впровадження. Апаратні ключі потребують додаткових витрат на закупівлю, налаштування та

подальше обслуговування. Для невеликих компаній або освітніх проєктів це може стати значним фінансовим бар’єром.

Другий недолік полягає у ризику фізичної втрати або пошкодження пристрою. Втрата апаратного ключа може призвести до блокування доступу до програмного забезпечення та необхідності звертатися до постачальника для відновлення ліцензії. Така ситуація особливо критична у фінансових або державних системах, де швидкий доступ до програмного забезпечення є обов’язковим для виконання робочих процесів.

Третім недоліком є вимога постійно мати доступний апаратний носій при собі. Це ускладнює роботу на декількох пристроях або у віддаленому режимі, що знижує мобільність користувачів і потребує додаткового планування робочого процесу.

Четвертий недолік пов’язаний із потенційною можливістю технічного обходу. Наприклад, у випадку спрощених методів прив’язки можлива емуляція апаратних ключів або підробка серійних номерів. Це ставить під питання абсолютну надійність системи, хоча для високоякісних реалізацій даний ризик мінімізується за рахунок поєднання апаратного ключа з криптографічними алгоритмами та багатофакторною автентифікацією.

1.4.3 Реальні сценарії використання.

Апаратна прив’язка активно використовується у корпоративних програмах для обмеження доступу співробітників до критичних модулів. Наприклад, бухгалтерські та ERP-системи часто використовують USB-ключі для підтвердження автентичності користувача, що дозволяє забезпечити багаторівневий контроль доступу та уникнути несанкціонованого доступу до фінансових даних і важливих бізнес-процесів. Таке рішення значно зменшує ризик внутрішніх загроз, коли доступ до системи може бути спробований навіть з боку співробітників компанії, які не мають відповідних повноважень.

У військових та державних установах апаратні ключі гарантують, що секретні дані не будуть використані на неавторизованих комп’ютерах,

забезпечуючи захист від витоків інформації та кібератак. Використання USB-ключів у таких системах дозволяє централізовано керувати доступом, вести облік спроб підключення та швидко реагувати на потенційні загрози. Крім того, апаратна прив’язка дозволяє інтегруватися з існуючими системами багатофакторної автентифікації, підвищуючи загальний рівень кібербезпеки організації.

У освітньому середовищі апаратні ключі дозволяють контролювати використання програмного забезпечення на комп’ютерах лабораторій та обмежувати доступ до ліцензованих продуктів лише для студентів і викладачів. Це гарантує, що програмне забезпечення не буде використано поза навчальним процесом або на особистих пристроях, що підвищує ефективність контролю за дотриманням ліцензійних умов та економить ресурси навчальних закладів.

У комерційних проєктах, де застосовується ліцензування на кількість робочих місць, апаратна прив’язка дозволяє ефективно управляти правами доступу та уникати зайвих витрат на додаткові ліцензії. Підприємства отримують можливість гнучко розподіляти ресурси між відділами, легко блокувати або перенаправляти доступ при зміні структури компанії та оперативно реагувати на потреби бізнесу.

Окрім того, апаратна прив’язка створює можливості для аналітики та моніторингу використання програмного забезпечення. Завдяки унікальним ідентифікаторам USB-ключів можна відстежувати активність користувачів, визначати типові патерни використання системи та оперативно виявляти аномальні спроби доступу, що дозволяє своєчасно реагувати на потенційні загрози та мінімізувати ризики.

Таким чином, апаратна прив’язка програмного забезпечення забезпечує високий рівень захисту та гнучкість у реалізації ліцензійних моделей, одночасно маючи обмеження, пов’язані з вартістю, фізичною безпекою та зручністю використання. Вибір цього методу залежить від специфіки програмного продукту, вимог до безпеки та бюджету. Для великих корпоративних, державних та фінансових систем апаратна прив’язка є оптимальною, тоді як у невеликих

комерційних або освітніх проєктах її застосування має бути обґрунтованим, продуманим та інтегрованим у загальну систему контролю доступу.

1.5 Постановка задачі

Метою магістерської роботи є підвищення ефективності захисту програмного з забезпечення від несанкціонованого використання шляхом розробки методу та засобу апаратної прив’язки до USB-носіїв на основі адаптивної схеми поділу секрету Шаміра динамічним керуванням порогом доступу та інтеграцією оцінки довіри (Trust Score) до факторів автентифікації.
Для досягнення цієї мети необхідно вирішити такі основні завдання:

1. Провести аналіз існуючих методів апаратної прив’язки програмного забезпечення до USB-носіїв та визначити їхні обмеження щодо масштабованості, гнучкості та рівня стійкості до компрометації.

2. Розробити адаптивну схему поділу секрету Шаміра, у якій ключ доступу до програмного забезпечення розподіляється на частини, що формуються на основі гетерогенних факторів автентифікації:

– USB-флешки (апаратний носій);

– Hardware ID (апаратний відбиток комп’ютера);

– пароля користувача (знання);

– одноразового коду TOTP (динамічний фактор).

3. Реалізувати механізм динамічного визначення порогу доступу (значення k) у схемі поділу секрету Шаміра залежно від контексту використання – часу доби, типу пристрою, геолокації та поведінкових патернів користувача. Наприклад, у звичних умовах (робочий комп’ютер, звичний час доби) система може вимагати лише два фактори автентифікації, тоді як у підозрілих ситуаціях (нова локація або нічний вхід) – три або більше.

4. Розробити алгоритм оцінки довіри (Trust Score) для кожного фактора автентифікації, який дозволяє системі адаптивно змінювати рівень безпеки. Система має “навчатися” на основі історії використання: часто застосовуваний

USB-ключ отримує вищу довіру, новий пристрій – нижчу, що впливає на підвищення або зниження порогу k.

5. Забезпечити можливість безпечного відновлення доступу до програмного забезпечення у випадку втрати USB-носія, використовуючи властивості схеми Шаміра для відновлення секрету через інші фактори (наприклад, пароль користувача + TOTP + recovery-ключ).

6. Розробити прототип програмного засобу, який реалізує запропонований метод захисту та забезпечує:

– автентифікацію користувача на основі кількох факторів;

– перевірку унікальності USB-пристрою та Hardware ID;

– адаптивне управління рівнем доступу залежно від контексту.

7. Провести експериментальну перевірку запропонованого методу, оцінити його ефективність за критеріями:

– здатність системи адаптуватися до поведінкових змін користувача;

– рівень зручності та швидкодії при реальному використанні.

1.6 Висновки з розділу

У першому розділі проведено аналіз сучасних загроз програмному забезпеченню, серед яких ключовими є несанкціоноване копіювання, реверс-інжиніринг, модифікація коду та запуск на непередбачених пристроях. Визначено, що сучасні атаки стають дедалі складнішими, тому для ефективного захисту потрібні комплексні рішення, що поєднують апаратні та програмні методи.

Розглянуто існуючі способи апаратної прив’язки, зокрема USB-донгли, прив’язку до серійних номерів носіїв, смарт-карти та криптотокени, а також комбіновані системи з мережевою авторизацією. Найбільш надійними є рішення з апаратними ключами, які підтримують криптографічні операції.

Окремо проаналізовано криптографічні методи захисту, зокрема AES, SHA-256 та схему поділу секрету Шаміра, які в комплексі із апаратною прив’язкою забезпечують високий рівень безпеки та контроль доступу.

Обґрунтовано вибір USB-носіїв як апаратного ідентифікатора завдяки їх унікальності, мобільності та неможливості простого копіювання. Вказано переваги та недоліки апаратної прив’язки, а також наведено реальні сценарії застосування в корпоративному, державному та освітньому секторах.

2 РОЗРОБКА МЕТОДУ ЗАХИСТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ПРИВ'ЯЗКОЮ ДО USB-НОСІЇВ

2.1 Концепція та архітектура методу

Розроблений метод базується на комбінуванні кількох гетерогенних факторів автентифікації в межах єдиної криптографічної схеми поділу секрету Шаміра з можливістю динамічної адаптації рівня безпеки. На відміну від традиційних підходів до апаратної прив'язки програмного забезпечення, які використовують статичну кількість факторів автентифікації, запропонований метод автоматично визначає необхідний рівень захисту на основі контекстної інформації та аналізу поведінкових патернів користувача.

Основна ідея методу полягає у розділенні головного ключа шифрування на *n* частин (shares) за допомогою схеми Шаміра, де кожна частина прив'язується до окремого фактора автентифікації. Для відновлення ключа та отримання доступу до захищеного програмного забезпечення необхідно надати k факторів з *n* можливих, при цьому значення порогу *k* не є фіксованим, а визначається динамічно на основі інтегральної оцінки довіри (Trust Score) до поточної спроби автентифікації.

У розробленому методі використовуються п'ять факторів автентифікації, кожен з яких представляє окремий рівень захисту:

– USB-токен як фізичний носій інформації.

– Hardware ID як прив'язка до конкретної апаратної конфігурації.

– Пароль користувача як фактор знання.

– TOTP-код як динамічний фактор часової синхронізації.

– Recovery-ключ як резервний механізм відновлення доступу.

Кожен фактор забезпечує незалежний канал верифікації ідентичності користувача, що підвищує загальний рівень криптографічної стійкості системи.

Призначення системи полягає у забезпеченні гнучкого багатофакторного захисту програмного забезпечення з автоматичною адаптацією рівня безпеки до поточних умов використання.

2.1.1 Архітектура системи

Архітектура запропонованого методу побудована за принципом багаторівневої модульності, що забезпечує масштабованість, розширюваність і можливість інтеграції з іншими системами автентифікації. Кожен модуль реалізує чітко визначену функцію в межах загальної логіки обробки факторів автентифікації, обчислення рівня довіри, формування порогу доступу та криптографічного відновлення секрету.

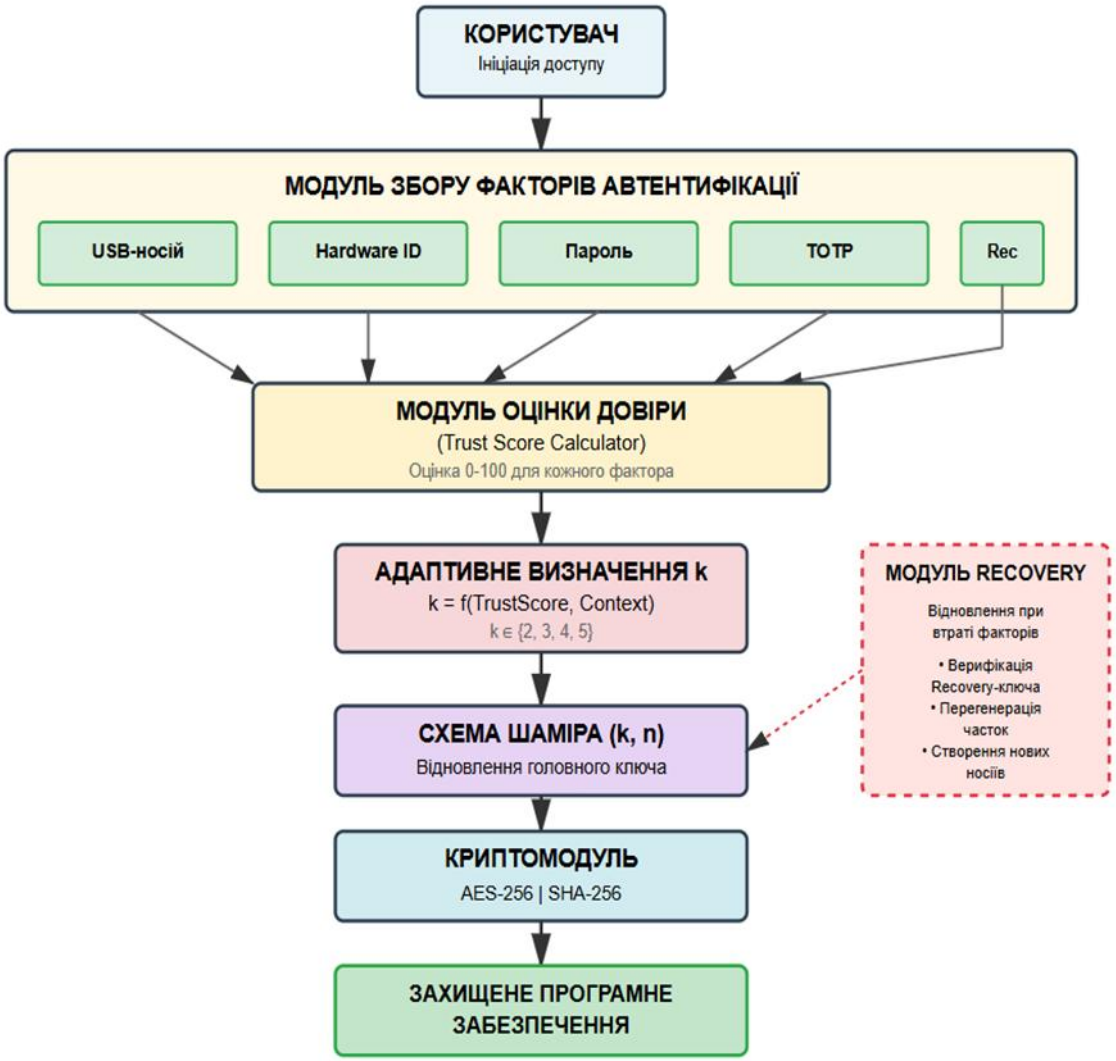


Рисунок 2.1 – Архітектура системи захисту програмного забезпечення

На рисунку 2.1 зображено архітектуру системи. Архітектура складається з шести основних модулів, що взаємодіють через чітко визначені інтерфейси.

1. Модуль збору факторів відповідає за отримання даних з USB-носія, збір апаратних характеристик, введення пароля та генерацію TOTP.
2. Модуль оцінки довіри аналізує зібрану інформацію та контекст для розрахунку Trust Score.
3. Модуль адаптивного визначення k використовує оцінки довіри для визначення необхідної кількості факторів.
4. Модуль схеми Шаміра реалізує криптографічний алгоритм поділу та відновлення секрету через поліноміальну інтерполяцію.
5. Криптомодуль забезпечує шифрування за алгоритмом AES-256, хешування SHA-256.
6. Модуль Recovery забезпечує відновлення доступу при втраті основних факторів автентифікації.

2.1.2 Пояснення потоків даних між модулями

Процес автентифікації починається з ініціації запиту користувачем. Модуль збору факторів визначає наявність USB-носія, зчитує Hardware ID, отримує пароль та TOTP-код. Паралельно збирається контекстна інформація: час доступу, IP-адреса, географічна локація. Зібрані дані передаються до модуля оцінки довіри.

Модуль оцінки довіри розраховує індивідуальні оцінки для кожного фактора та загальну оцінку сесії. Результати передаються до модуля адаптивного визначення *k*, який приймає рішення про необхідну кількість факторів. При достатній кількості факторів модуль Шаміра відновлює головний ключ, який передається до криптомодуля для розшифрування захищених компонентів.

2.2 Формування та розподіл секрету за схемою Шаміра

Схема поділу секрету Шаміра є криптографічним методом, що дозволяє розділити секретну інформацію на n частин таким чином, що для відновлення секрету необхідно мати щонайменше k частин. У розробленому методі схема Шаміра використовується для розподілу головного ключа доступу між п'ятьма факторами автентифікації, забезпечуючи гнучкість та високий рівень захисту.

2.2.1 Математичні основи схеми Шаміра

Схема Шаміра базується на властивості поліноміальної інтерполяції: для однозначного визначення поліному степеня (k-1) необхідно мати щонайменше k точок. Нехай S – секрет, який потрібно розділити. Обирається просте число p, більше за S та за максимальне значення n. Формується поліном степеня (k-1):

$$f(x) = a^0 + a^1x + a^2x^2 + \ldots + a_{(k-1)x}^{k-1}(mod\ p), \quad (2.1)$$

де $a_0 = S$ – секрет;

$a_1, a_2, \ldots, a\_(k-1)$ – випадкові коефіцієнти з діапазону $[0, p-1]$;

p – просте число [біт].

Для кожного учасника i (де i = 1, 2, ..., n) обчислюється частка секрету як пара (i, $f(i)$). Кожна частка передається відповідному фактору автентифікації. Ключовою властивістю схеми є те, що володіння менше ніж k частками не надає жодної інформації про секрет S. З математичної точки зору, через будь-які менше ніж k точок можна провести нескінченну кількість поліномів степеня (k-1), що робить неможливим визначення правильного поліному без достатньої кількості часток.

Для відновлення секрету використовується інтерполяційний поліном Лагранжа. Маючи k точок (x_1, y_1), (x_2, y_2), ..., (x_k, y_k), секрет відновлюється як значення поліному в точці x=0:

$$S = f(0) = \Sigma(i = 1\ to\ k)y_{\text{i}} \cdot \Pi(j = 1\ to\ k, j \neq i)\left(\frac{x_{\text{j}}}{x_{\text{j}} - x_{\text{i}}}\right)(mod\ p), \quad (2.2)$$

де S – відновлений секрет;

k – порогове значення часток;

y_{i} – значення і-тої частки;

$x_{\text{j}}, x_{\text{i}}$ – ідентифікатори часток;

p – просте число [біт].

У розробленій системі використовується схема (k, 5), де n=5 відповідає п'яти факторам автентифікації, а k є змінним параметром, що визначається динамічно в діапазоні від 2 до 5. Для забезпечення криптографічної стійкості використовується просте число p розміром 512 біт, що значно перевищує розмір головного ключа (256 біт) та забезпечує достатній запас для безпечних обчислень.

2.2.2 Процес генерації та розподілу часток секрету

При ініціалізації системи захисту виконується процедура генерації головного ключа та його розподілу між факторами автентифікації. Цей процес є критично важливим для безпеки системи, оскільки від якості генерації випадкових коефіцієнтів залежить стійкість всієї схеми до криптоаналізу.

Перед початком генерації система проводить ініціалізацію середовища криптографічної безпеки, що включає перевірку доступності джерел ентропії, цілісності бібліотек та коректності роботи модуля криптографічних операцій. Далі створюється сесійний контекст, у якому фіксуються параметри користувача, версія алгоритмів і час генерації ключа. У цьому контексті формується набір параметрів для поділу секрету: кількість часток n = 5, порогове значення k (залежно від політики безпеки), а також вибір типів факторів автентифікації, між якими буде розподілено секрет. На рисунку 2.2 зображено алгоритм генерації та розподілу часток секрету.

Рисунок 2.2 – Алгоритм генерації та розподілу часток секрету

Генерація починається зі створення головного ключа S – випадкового 256-бітного значення, сформованого криптостійким ГПВЧ (CSPRNG). Для обчислень обирається просте число p розміром 512 біт, що забезпечує достатній рівень безпеки. Далі генеруються чотири випадкові коефіцієнти $a1 \dots a4 \in [0, p-1]$ відповідні максимальному порогу $kmax = 5$, після чого формується поліном четвертого степеня:

$$f(x) = S + a^1 x + a^2 x^2 + a^3 x^3 + a^4 x^4 (mod\ p), \quad (2.3)$$

де S – головний ключ [біт];

a_1, a_2, a_3, a_4 – випадкові коефіцієнти;

x – ідентифікатор фактора;

p – просте число [біт].

Для кожного з п'яти факторів автентифікації обчислюється відповідна частка секрету. Факторам присвоюються унікальні ідентифікатори від 1 до 5: USB-носій отримує ідентифікатор 1, Hardware ID – ідентифікатор 2, пароль – ідентифікатор 3, TOTP – ідентифікатор 4, Recovery-ключ – ідентифікатор 5. Для кожного ідентифікатора і обчислюється значення поліному *f(i)* за модулем *p*, що формує частку *(i, f(i))*. Розподіл часток між факторами автентифікації та місця їх зберігання наведено в таблиці 2.1.

Обчислені частки зберігаються відповідно до типу фактора. USB-частка записується в прихований розділ носія й додатково шифрується ключем, прив’язаним до серійного номера пристрою.

1. Частка для Hardware ID зберігається у системі в зашифрованому вигляді, а ключ формується на основі унікальних апаратних ідентифікаторів.
2. Частка пароля не зберігається: при введенні пароля обчислюється SHA-256 хеш, який використовується для розшифрування зашифрованої частки.
3. Частка TOTP розміщується у захищеному сховищі разом із секретним ключем генерації кодів.

4. Recovery-ключ зберігається окремо у безпечному вигляді (паперовий документ, QR-код).

Таблиця 2.1 – Розподіл часток секрету між факторами автентифікації

Частина	Фактор автентифікації	Місце зберігання	Спосіб захисту
1	USB-носій	Зашифрований файл на захищеному розділі флешки	Шифрування з прив'язкою до серійного номера USB
2	Hardware ID	Локально в системному реєстрі або конфігураційному файлі	Шифрування на основі апаратних характеристик
3	Пароль користувача	Зашифрована частка в базі даних системи	Шифрування ключем, похідним від хешу пароля
4	TOTP	Seed у захищеному контейнері на сервері	Доступ після валідації поточного TOTP-коду
5	Recovery Key	Резервна частина поза системою	Зберігається окремо, використовується для відновлення

Для перевірки коректності відновлення ключа зберігається контрольна сума $SHA-256(S)$.Після завершення генерації всі дані про секрет і коефіцієнти полінома безпечно очищуються з пам’яті.

2.2.3 Алгоритм відновлення секрету

При спробі доступу до захищеного програмного забезпечення виконується процес відновлення головного ключа на основі доступних часток від факторів автентифікації. Успішність відновлення залежить від наявності достатньої

кількості валідних часток, що визначається пороговим значенням k. На рисунку 2.3 зображено алгоритм відновлення секрету за схемою Шаміра.

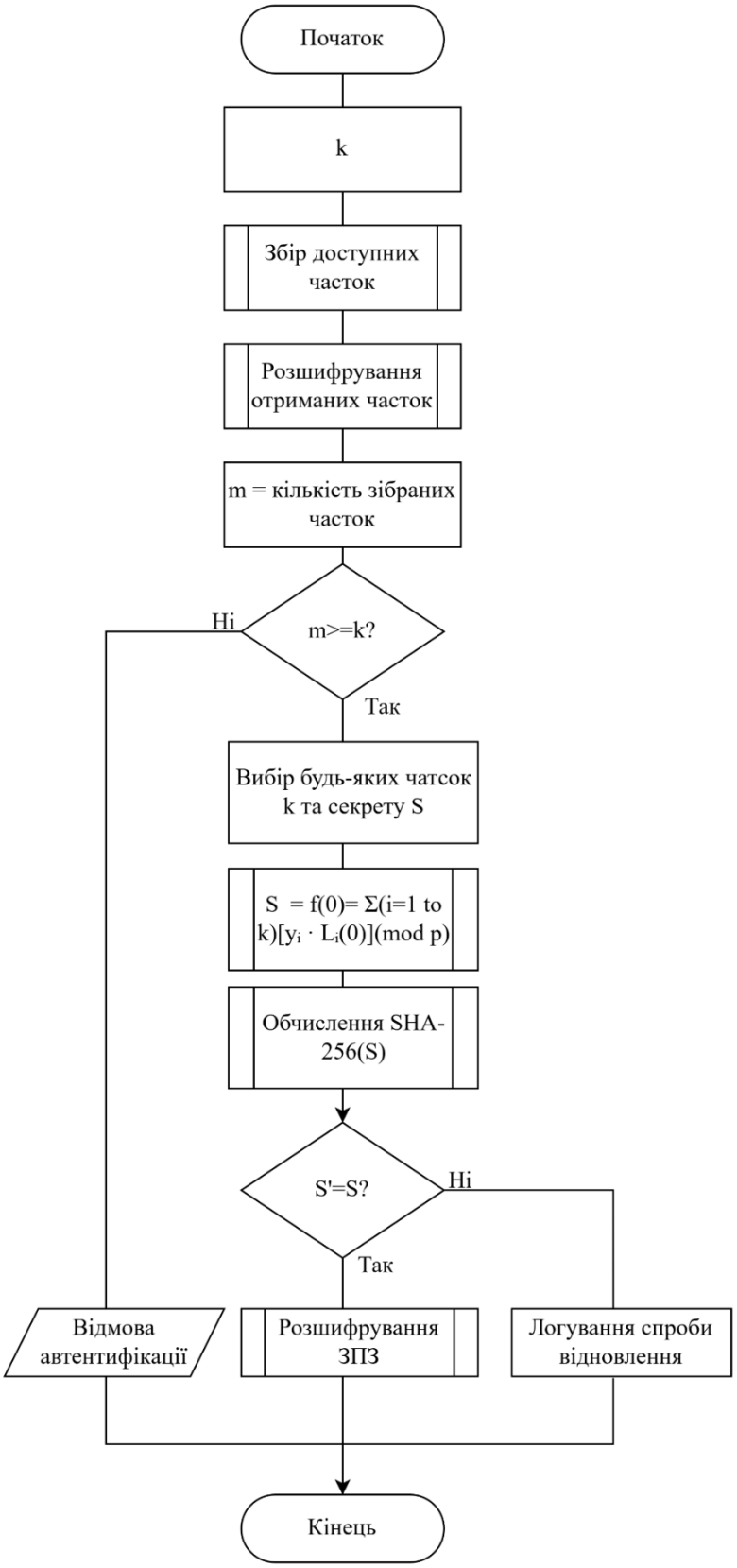


Рисунок 2.3 – Алгоритм відновлення секрету за схемою Шаміра

Процес відновлення починається з отримання порогу k, після чого модуль збору факторів намагається отримати частки зі всіх доступних джерел. Частки з USB та Hardware ID розшифровуються ключами, прив'язаними відповідно до носія та конфігурації обладнання. Частка пароля стає доступною після успішної перевірки SHA-256 хешу введеного пароля, частка TOTP – після підтвердження одноразового коду.

Підраховується кількість наявних часток m. Якщо $m < k$, автентифікація завершується відмовою. Якщо $m \geq k$, обираються будь-які k часток і секрет S відновлюється за інтерполяцією Лагранжа як значення полінома в точці $x = 0$, що відображено у формулі (2.2).

Обчислення виконуються за модулем p з використанням модулярної арифметики великих чисел. Важливо забезпечити коректність обчислень зворотних елементів при діленні за модулем, що досягається використанням розширеного алгоритму Евкліда.

Після відновлення значення S система обчислює його контрольну суму за алгоритмом SHA-256 і порівнює з раніше збереженим хешем. Якщо контрольні суми співпадають, відновлення вважається успішним і ключ S передається до криптомодуля для розшифрування захищеного програмного забезпечення. Збіг контрольних сум підтверджує не тільки правильність математичних обчислень, але й те, що були надані валідні частки від легітимних факторів автентифікації.

Якщо контрольні суми не співпадають, це свідчить про помилку відновлення, що може бути спричинена пошкодженням даних у сховищах часток, використанням невалідних факторів автентифікації або помилками в обчисленнях. У такому випадку система відмовляє в доступі і логує спробу з деталями помилки.

2.3 Модель оцінки довіри

Модель оцінки довіри є ключовим компонентом адаптивного методу захисту, що дозволяє системі динамічно визначати рівень безпеки на основі аналізу поведінки користувача та контексту автентифікації. Trust Score – це числовий показник в діапазоні від 0 до 100 балів, який відображає ступінь довіри системи до поточної спроби доступу.

2.3.1 Сутність показника довіри

Основна ідея Trust Score полягає в тому, що не всі спроби автентифікації є рівноцінними з точки зору ризику. Доступ з офісного комп'ютера в робочий час з використанням звичного USB-носія має високий рівень довіри. Натомість, спроба доступу о третій ночі з незнайомої IP-адреси після кількох невдалих спроб викликає підозру і вимагає посилених заходів безпеки.

Trust Score обчислюється на основі чотирьох основних факторів: статусу USB-носія (відомий/новий), часу входу (нормальний/аномальний), геолокації та IP-адреси, історії попередніх входів та кількості невдалих спроб. Кожен фактор отримує нормалізовану оцінку від 0 до 1, де 1 означає максимальну довіру, а 0 – максимальну підозру.

2.3.2 Фактори оцінки довіри

1. Фактор USB-носія (F_{usb}). Цей фактор оцінює, чи є USB-носій відомим системі і як часто він використовується. Якщо USB-носій використовується регулярно (щодня або щотижня), $F_{usb} = 1,0$. Якщо носій новий, але успішно пройшов перевірку – $F_{usb} = 0,5$. Якщо носій відсутній або невідомий – $F_{usb} = 0,3$. При виявленні підозрілих ознак (невідповідність серійного номера) – $F_{usb} = 0$.
2. Фактор часу входу (F_{time}). Система аналізує, чи відповідає час входу звичному графіку користувача. Вхід в робочі години (для офісного працівника це зазвичай 9:00-18:00) дає $F_{time} = 1,0$. Вхід у позаробочий час, але в межах звичного діапазону (наприклад, 8:00 або 19:00) – $F_{time} = 0,7$.

Вхід в нічні години (22:00-6:00) для користувача, що зазвичай працює вдень – $F_{time} = 0,2$. Система накопичує статистику і визначає типові години активності індивідуально для кожного користувача.

3. Фактор геолокації ($F_{location}$). Визначається на основі IP-адреси та географічного розташування. Вхід з офісної мережі або домашньої IP-адреси дає $F_{location} = 1,0$. Вхід з нової локації в тому ж місті – $F_{location} = 0,6$. Вхід з іншого міста або країни – $F_{location} = 0,3$. Система також перевіряє фізичну можливість переміщення: якщо попередній вхід був з Києва 30 хвилин тому, а поточний – з Лондона, $F_{location} = 0$.
4. Фактор історії спроб ($F_{attempts}$). Враховує кількість невдалих спроб автентифікації за останні 15 хвилин. Відсутність невдалих спроб – $F_{attempts} = 1,0$. Одна невдала спроба – $F_{attempts} = 0,8$. Дві спроби – $F_{attempts} = 0,5$. Три і більше – $F_{attempts} = 0,2$. При перевищенні порогу (5 спроб) обліковий запис тимчасово блокується.

2.3.3 Формула розрахунку Trust Score

Загальний показник довіри обчислюється як зважена сума чотирьох факторів:

$$Trust = 0,4 \cdot Fusb + 0,3 \cdot Ftime + 0,2 \cdot Flocation + 0,1 \cdot Fattempts,$$

де $Trust$ – загальний показник довіри [0..1];

$Fusb$ – оцінка USB-носія [0..1];

$Ftime$ – оцінка часу входу [0..1];

$Flocation$ – оцінка геолокації [0..1];

$Fattempts$ – оцінка історії спроб [0..1].

Для перетворення в діапазон 0-100 балів використовується множення на 100:

$$TrustScore = Trust \times 100.$$

Вагові коефіцієнти факторів Trust Score визначено аналітичним шляхом на основі порівняльного аналізу впливу кожного фактора на загальний рівень безпеки системи автентифікації.

Аналіз проведено з урахуванням критеріїв:

- складність компрометації фактора (ймовірність підробки або крадіжки);
- стабільність значення фактора у часі;
- можливість автоматичної перевірки достовірності;
- вплив на ризик несанкціонованого доступу при реальних сценаріях використання.

Вагові коефіцієнти та їх обґрунтування наведено в таблиці 2.2.

Таблиця 2.2 – Вагові коефіцієнти факторів Trust Score

№	Критерій оцінки	Ймовірність компрометації	Складність перевірки достовірності	Вплив на рівень безпеки	Вага (частка)
1	USB-носій	Фізичний доступ, унікальний серійний номер	Дуже низька	Висока (перевірка через HWID/серійний ID)	0,4 (40%)
2	Час входу	Відповідність типовому графіку користувача	Середня	Висока (автоматичний збір)	0,3 (30%)
3	Геолокація	Відповідність звичним IP/містам	Середня	Середня	0,2 (20%)
4	Історія спроб	Кількість невдалих входів	Висока	Висока (логування)	0,1 (10%)

2.3.4 Приклад зміни Trust Score у часі

Розглянемо типовий сценарій роботи користувача Івана, який працює в офісі з понеділка по п'ятницю з 9:00 до 18:00. Зміну Trust Score протягом тижня наведено в таблиці 2.3.

Таблиця 2.3 – Приклад зміни Trust Score у типових сценаріях

Сценарій	F_{usb}	F_{time}	$F_{location}$	$F_{attempts}$	Trust	Інтерпретація
Типовий робочий день (10:00, офіс)	1,0	1,0	1,0	1,0	1,0	Висока довіра (100)
Ранній вхід (7:30, офіс)	1,0	0,7	1,0	1,0	0,91	Висока довіра (91)
Робота вдома (14:00)	1,0	1,0	0,6	1,0	0,92	Висока довіра (92)
Вхід без USB (10:00, офіс)	0,3	1,0	1,0	1,0	0,62	Середня довіра (62)
Відрядження (11:00, інше місто)	1,0	1,0	0,3	1,0	0,76	Середня довіра (76)
Нічний вхід (2:00, дім)	1,0	0,2	0,6	1,0	0,58	Знижена довіра (58)
Після 2 невдалих спроб (10:00, офіс)	1,0	1,0	1,0	0,5	0,95	Висока довіра (95)
Підозріла активність (3:00, нова IP, 3 невдалі спроби)	0,3	0,2	0,3	0,2	0,26	Низька довіра (26)

З таблиці 2.3 видно, як система реагує на різні сценарії. Типова робота в офісі дає максимальну оцінку 100 балів. Незначні відхилення (ранній вхід,

робота вдома) знижують оцінку незначно – до 91-92 балів, що все ще є високою довірою. Відсутність USB-носія знижує оцінку до 62 балів (середня довіра), що вимагатиме додаткових факторів автентифікації. Підозріла активність (нічний вхід з нової IP після невдалих спроб) дає тільки 26 балів, що вимагає максимальних заходів безпеки.

2.3.5 Поведінкове навчання системи

Ключовою особливістю моделі Trust Score є здатність системи до самонавчання та адаптації під індивідуальні патерни поведінки користувача. Система не використовує фіксовані правила, а накопичує статистику про кожного користувача і з часом підвищує точність оцінок.

Накопичення профілю користувача. Для кожного користувача система веде профіль, що містить: розподіл часу входу по годинах доби (гістограма активності), список звичних IP-адрес та локацій з частотою використання, історію використання USB-носіїв, середню тривалість робочих сесій. Профіль формується на основі перших 20-30 успішних сесій автентифікації і потім постійно оновлюється.

Визначення типових патернів. На основі накопичених даних система виявляє регулярні патерни. Наприклад, якщо Іван завжди входить в систему з 9:00 до 9:30 з офісної IP, це формує патерн "ранковий вхід". Якщо він також часто працює ввечері (18:00-19:00) з домашньої IP, формується другий патерн "вечірня робота". Ці патерни використовуються для оцінки фактора Ftime та Flocation.

Адаптація до змін. Коли користувач змінює свою поведінку (наприклад, переходить на віддалену роботу), система не відразу довіряє новому патерну. Перші входи з домашньої IP отримують знижену оцінку Flocation = 0,6. Після 5-7 успішних входів з тієї ж локації система починає розпізнавати новий патерн і поступово підвищує оцінку до 0,9-1,0. Цей механізм забезпечує баланс між безпекою та зручністю.

Виявлення аномалій. Система порівнює поточну сесію з усіма накопиченими даними. Якщо виявляється поведінка, що ніколи раніше не спостерігалася (наприклад, вхід з нової країни), це розглядається як аномалія і знижує Trust Score. Чим більше накопичено даних, тим точніше система виявляє аномалії. Для нового користувача (менше 10 сесій) система працює в режимі "навчання" з підвищеними вимогами до безпеки.

Механізм забування. Старі дані поступово втрачають вагу. Записи старіші за 6 місяців отримують знижену вагу при обчисленні статистики, а записи старіші за рік можуть бути архівовані. Це дозволяє системі адаптуватися до довгострокових змін в поведінці користувача (наприклад, зміна місця роботи, переїзд в інше місто).

Приклад адаптації. Розглянемо, як змінюється оцінка Flocation при переході на віддалену роботу:

День 1 (перший вхід з дому): Flocation = 0,3 (нова локація) → Trust = 74 бали;

День 2-3: Flocation = 0,5 (система фіксує повторення) → Trust = 80 балів;

День 4-7: Flocation = 0,7 (формується новий патерн) → Trust = 88 балів;

День 8+: Flocation = 0,9-1,0 (новий патерн підтверджено) → Trust = 96-100 балів.

Така поступова адаптація забезпечує, що система не блокує легітимного користувача при зміні умов роботи, але водночас залишається пильною до справді підозрілої активності.

Таким чином, модель оцінки довіри забезпечує інтелектуальний механізм визначення рівня безпеки, що враховує як поточні параметри автентифікації, так і історичну поведінку користувача. Здатність до самонавчання дозволяє системі адаптуватися під індивідуальні особливості кожного користувача, підвищуючи як безпеку, так і зручність використання.

2.4 Адаптивне визначення порогу k

Адаптивне визначення порогу k є ключовим механізмом, що забезпечує баланс між безпекою та зручністю використання системи. На відміну від традиційних методів з фіксованими вимогами до автентифікації, розроблений метод динамічно змінює кількість необхідних факторів залежно від оцінки довіри Trust Score та поточного контексту.

2.4.1 Принцип адаптивної зміни кількості факторів

Основна ідея адаптивного визначення k полягає в тому, що система автоматично підвищує вимоги до автентифікації при виявленні підозрілих ознак та знижує їх при типовій роботі користувача. Це дозволяє легітимним користувачам працювати з мінімальними перешкодами в звичних умовах, але суттєво ускладнює несанкціонований доступ.

Поріг k визначається на основі розрахованого значення Trust Score згідно з наступними правилами:

Trust Score ≥ 80 балів → k = 2. Висока довіра. Користувач працює в звичних умовах (типовий час, локація, USB-носій). Достатньо двох факторів, наприклад: USB + TOTP або HWID + TOTP.

Trust Score від 50 до 79 балів → k = 3. Середня довіра. Виявлено незначні відхилення від норми (нестандартний час або нова локація). Потрібно три фактори: USB + пароль + TOTP або USB + HWID + пароль.

Trust Score від 30 до 49 балів → k = 4. Знижена довіра. Виявлено суттєві відхилення або аномалії. Потрібно чотири фактори: USB + HWID + пароль + TOTP.

Trust Score < 30 балів → k = 5. Критично низька довіра. Множинні аномалії або висока ймовірність атаки. Вимагаються всі п'ять факторів, включно з Recovery-ключем.

2.4.2 Алгоритм вибору порогу k

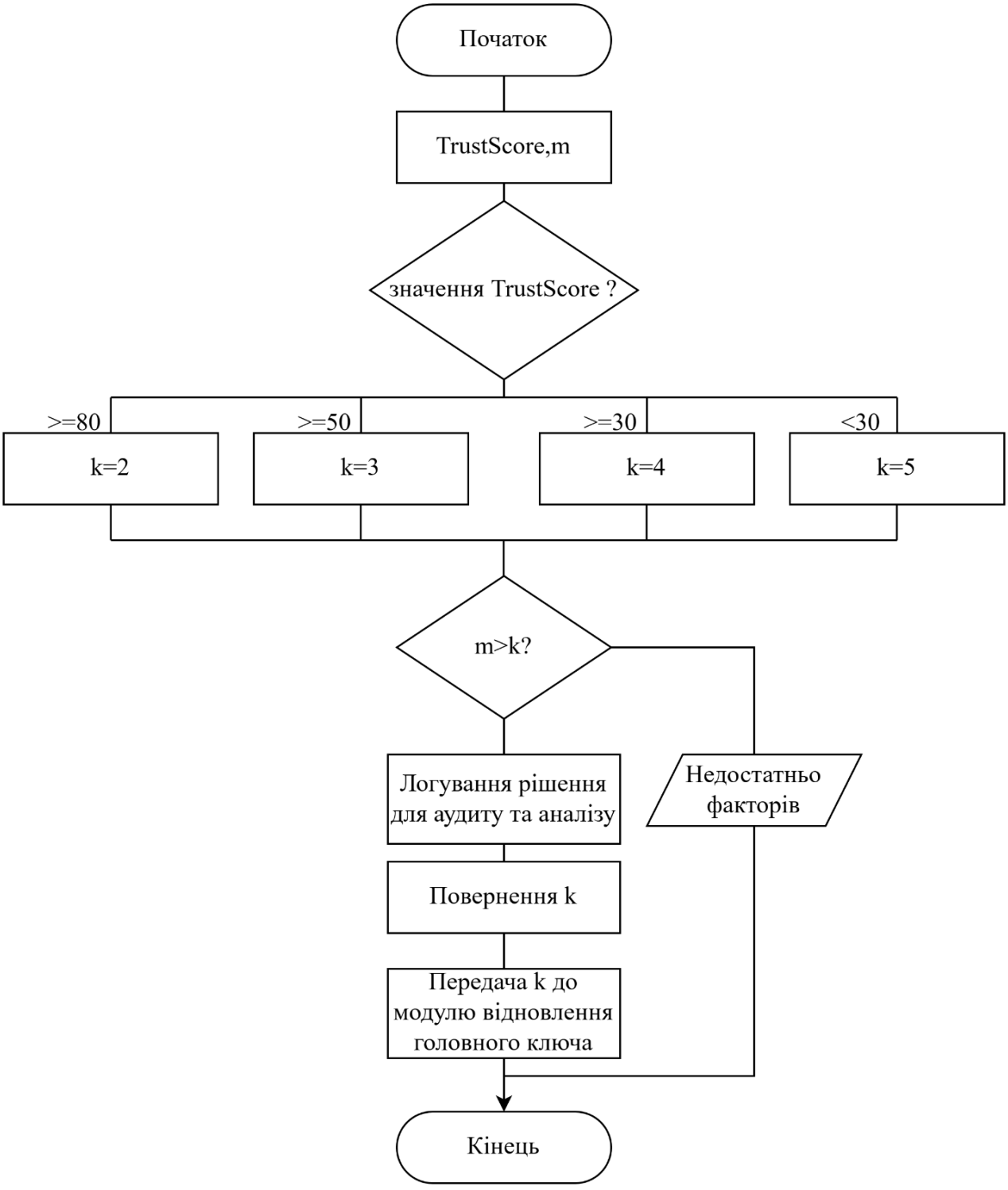


Рисунок 2.4 – Алгоритм адаптивного визначення порогу k

Опис роботи алгоритму, який зображено на рисунку 2.4:

1. Початок роботи. Алгоритм отримує два вхідні параметри: TrustScore (оцінка довіри від 0 до 100) та m (кількість наданих користувачем факторів).

2. Перевірка TrustScore ≥ 80. Якщо оцінка довіри висока (80 і більше балів), встановлюється мінімальний поріг k=2. Це означає, що користувач працює в типових умовах і достатньо двох факторів автентифікації.

3. Перевірка TrustScore ≥ 50. Якщо оцінка від 50 до 79, встановлюється k=3. Виявлено незначні відхилення від звичної поведінки, потрібен додатковий фактор.

4. Перевірка TrustScore ≥ 30. При оцінці від 30 до 49 балів встановлюється k=4. Система зафіксувала суттєві відхилення або аномалії.

5. Встановлення k=5. Якщо TrustScore < 30, встановлюється максимальний поріг k=5. Критично низька довіра вимагає всіх п'яти факторів автентифікації.

6. Перевірка достатності факторів. Система порівнює кількість наданих факторів m з необхідним k. Якщо m < k, генерується помилка "Недостатньо факторів" і користувачу пропонується надати додаткові фактори автентифікації.

7. Логування та завершення. При успішному визначенні k система логує рішення (TrustScore, обране k, час) для аудиту та аналізу. Алгоритм повертає значення k, яке передається модулю схеми Шаміра для відновлення головного ключа.

2.4.3 Приклади сценаріїв автентифікації

Розглянемо типові сценарії використання системи та відповідні значення k. Приклади наведено в таблиці 2.4.

Таблиця 2.4 – Сценарії автентифікації та визначення порогу k

Сценарій	Умови	Trust Score	k	Необхідні фактори
Звичайний робочий день	10:00, офіс, свій USB, без невдалих спроб	100	2	USB + TOTP
Ранній вхід	7:30, офіс, свій USB	91	2	USB + TOTP
Робота вдома	14:00, домашня IP (зареєстрована), свій USB	92	2	USB + TOTP
Забутий USB	10:00, офіс, без USB	62	3	HWID + Пароль + TOTP
Відрядження	11:00, інше місто (перший день), свій USB	76	3	USB + Пароль + TOTP
Нічний доступ	2:00, дім, свій USB	58	3	USB + Пароль + TOTP
Нічний доступ без USB	3:00, дім, без USB	46	4	HWID + Пароль + TOTP + Recovery
Аномальний вхід	3:00, нова IP, без USB, 2 невдалі спроби	26	5	Всі 5 факторів + блокування

2.4.4 Діаграма станів системи

Система адаптивного визначення порогу k може перебувати в чотирьох основних станах, що відповідають різним рівням безпеки. Переходи між станами

відбуваються автоматично на основі зміни Trust Score. Діаграму станів наведено на рисунку 2.5.

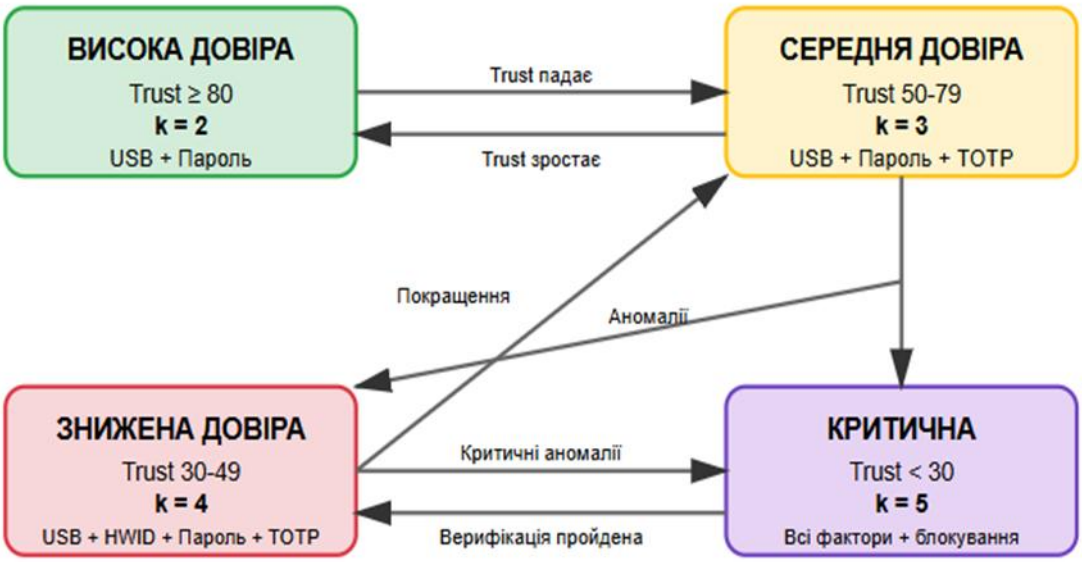


Рисунок 2.5 – Діаграма станів системи адаптивного визначення порогу k

Діаграма показує, що система може перебувати в одному з чотирьох станів, кожен з яких характеризується своїм значенням k та вимогами до факторів автентифікації. Переходи між станами відбуваються автоматично при зміні Trust Score. Наприклад, якщо користувач працює в стані "Висока довіра" (k=2) і раптово змінює локацію на незвичну, Trust Score падає і система переходить в стан "Середня довіра" (k=3), вимагаючи додатковий фактор TOTP.

Важливою особливістю системи є можливість повернення до менш обмежувальних станів. Якщо користувач успішно пройшов посилену автентифікацію і продовжує працювати нормально, Trust Score поступово зростає і система може повернутися до стану з нижчим k. Це забезпечує, що тимчасові відхилення не призводять до постійного ускладнення роботи.

Таким чином, адаптивне визначення порогу k забезпечує гнучкий механізм управління рівнем безпеки, що автоматично адаптується до поточної ситуації.

2.5 Алгоритм автентифікації користувача

Алгоритм автентифікації користувача об'єднує всі розроблені компоненти системи в єдиний процес перевірки легітимності доступу до захищеного програмного забезпечення.

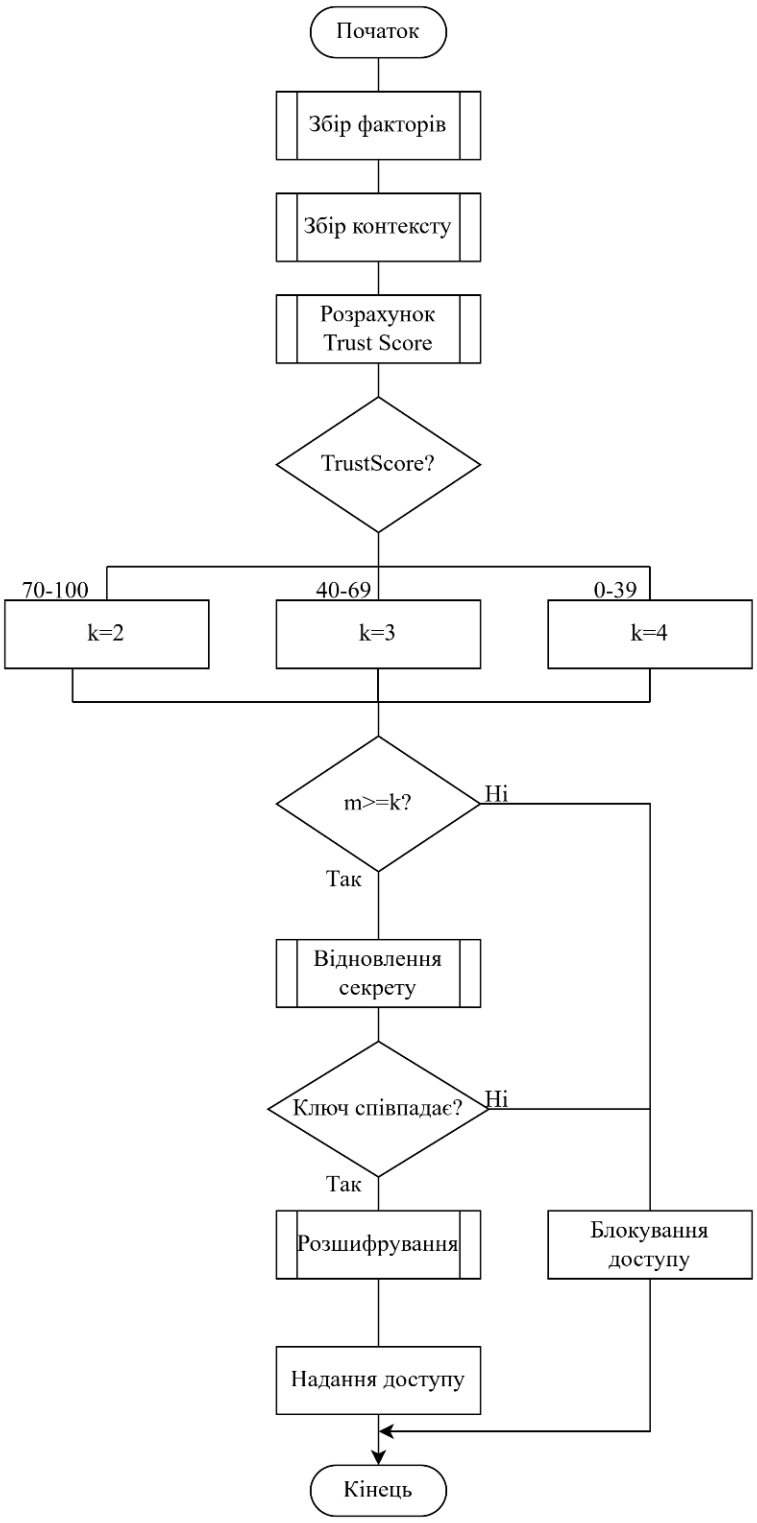


Рисунок 2.6 – Алгоритм автентифікації користувача

Процес включає збір факторів автентифікації, обчислення оцінки довіри, адаптивне визначення порогу k, відновлення секрету за схемою Шаміра та надання або блокування доступу, що зображено на рисунку 2.6.

1. Запит доступу. Користувач запускає захищене програмне забезпечення. Система ініціює автентифікацію та фіксує час запиту, IP-адресу та параметри сесії.

2. Збір факторів. Модуль автоматично перевіряє USB-носій та зчитує Hardware ID, запитує пароль через захищений інтерфейс та TOTP-код з мобільного додатку. Підраховується кількість наданих факторів m.

3. Збір контексту. Система збирає поточний час, день тижня, IP-адресу, географічну локацію через GeoIP, тип мережі та кількість невдалих спроб за останні п'ятнадцять хвилин.

4. Розрахунок Trust Score. Система обчислює оцінку довіри, аналізуючи якість факторів та контекст. Наприклад, щоденний USB-носій, робочі години, офісна IP та відсутність невдалих спроб дають TrustScore = 100 балів.

5. Визначення k. На основі TrustScore встановлюється необхідна кількість факторів: при 80-100 балах k=2, при 50-79 балах k=3, при 30-49 балах k=4, при 0-29 балах k=5.

6. Перевірка достатності. Якщо $m < k$, доступ блокується з повідомленням про необхідні фактори. Користувач може надати додаткові фактори або ініціювати Recovery. Якщо $m \geq k$, процес продовжується.

7. Відновлення секрету. Модуль Шаміра відновлює головний ключ S з k наданих факторів за інтерполяцією Лагранжа. Обчислюється SHA-256(S) і порівнюється зі збереженою контрольною сумою. При співпадінні ключ передається до криптомодуля.

8. Розшифрування. Криптомодуль розшифровує компоненти програми за AES-256-GCM. Після успішного розшифрування система надає доступ, ініціалізує сесію та логує автентифікацію з деталями.

2.6 Генерація, зберігання та оновлення ключів

Управління криптографічними ключами в розробленій системі охоплює повний життєвий цикл — від генерації головного ключа до його безпечного знищення. Генерація головного ключа S виконується під час реєстрації користувача з використанням криптографічно стійкого генератора псевдовипадкових чисел. Формується 256-бітний ключ, що відповідає вимогам стійкості алгоритму AES-256.

Подальший розподіл головного ключа між факторами автентифікації здійснюється за пороговою схемою Шаміра, математичні основи якої наведено у підрозділі 2.2. У системі використовується схема (k,5) з максимальним порогом k_{max}=5, де п’ять часток відповідають п’яти незалежним факторам автентифікації. Конкретне порогове значення k визначається динамічно відповідно до політики безпеки та рівня довіри (Trust Score).

У результаті застосування схеми формується п’ять часток секрету $Share_1$…$Share_5$, кожна з яких асоційована з окремим фактором автентифікації: USB-носієм, Hardware ID, паролем, TOTP та Recovery-ключем. Кожна частка зберігається окремо та додатково захищається криптографічними методами.

Частка $Share_1$ шифрується алгоритмом AES-256-GCM з ключем, похідним від хешу SHA-256 апаратних характеристик USB-пристрою, та зберігається у прихованому файлі на токені. Частка $Share_2$ зберігається у системному сховищі операційної системи та шифрується ключем, отриманим з Hardware ID. Частка $Share_3$ зберігається на сервері та захищається за допомогою PBKDF2-HMAC-SHA256 з використанням пароля користувача, сілі та 100 000 ітерацій. Частка $Share_4$ шифрується ключем, похідним від TOTP seed. Частка $Share_5$ захищається ключем, отриманим з мнемонічної фрази через PBKDF2 з 50 000 ітерацій. Параметри зберігання часток наведено в таблиці 2.6.

Відновлення доступу до системи здійснюється шляхом інтерполяції Лагранжа відповідно до схеми Шаміра. Мінімальна кількість часток, необхідних для відновлення, визначається динамічно. У режимі відновлення доступу

користувач повинен надати Recovery-ключ та щонайменше один додатковий фактор автентифікації, що забезпечує досягнення порогу k=2 та унеможливлює відновлення секрету при компрометації одного фактора.

Оновлення ключів реалізовано у трьох сценаріях: планова ротація кожні 90 днів із генерацією нового головного ключа та повторним розподілом часток; часткове оновлення при зміні окремих факторів автентифікації; а також відкликання та перешифрування часток у разі заміни апаратних компонентів. Безпечне видалення застарілих ключових матеріалів виконується шляхом багаторазового перезапису відповідно до стандарту DoD 5220.22-M.

Таблиця 2.6 – Параметри зберігання частин секрету

Частина	Розташування	Метод шифрування
$Share_1$	USB-токен	AES-256-GCM, ключ = SHA-256(SerialNumber||VendorID||ProductID)
$Share_2$	Реєстр/файл ОС	AES-256-GCM, ключ = SHA-256(CPU||MB||MAC||UUID)
$Share_3$	Сервер БД	AES-256-GCM, ключ = PBKDF2(Password, Salt, 100k iter)
$Share_4$	Сервер БД	AES-256-GCM, ключ = HMAC-SHA256(TOTP_seed)
$Share_5$	Сервер БД	AES-256-GCM, ключ = PBKDF2(Mnemonic24, Salt, 50k iter)

Термінове оновлення при компрометації включає відкликання поточних ключів, завершення всіх сесій, генерацію нового набору, повторну реєстрацію факторів з верифікацією через альтернативні канали.

2.7 Модуль відновлення доступу

Модуль відновлення забезпечує відновлення доступу при втраті факторів автентифікації зі збереженням високого рівня безпеки. Користувач ініціює процедуру через інтерфейс "Відновлення доступу", вводить ідентифікатор або email. Система відображає інформацію про зареєстровані фактори без конфіденційних деталей: останні чотири цифри серійного номера USB, частину Hardware ID, дату зміни пароля.

Користувач надає Recovery-ключ у формі мнемонічної фрази з двадцяти чотирьох слів за стандартом BIP39. Система валідує через контрольну суму та перевірку словника. З валідної мнемоніки виводиться ключ через PBKDF2-HMAC-SHA512 з сіллю "mnemonic". Ключ використовується для дешифрування $Share_5$ з бази даних.

Додатково користувач надає щонайменше один інший фактор, типово пароль або TOTP-код, для досягнення $k = 2$. Це захищає від несанкціонованого відновлення при компрометації Recovery-ключа. Система виконує автентифікацію з наданими факторами та відновлює головний ключ за схемою Шаміра через інтерполяцію Лагранжа.

При успішному відновленні користувач оновлює втрачені фактори. Для нового USB генерується $Share_1$ з тим самим значенням полінома, шифрується для нового пристрою, записується на токен, старий відкликається. При зміні комп'ютера генерується новий Hardware ID від поточної конфігурації, перешифровується $Share_2$. Для нового пароля система вимагає подвійного введення, перевіряє складність, генерує сіль, перешифровує $Share_3$. При компрометації TOTP генерується новий seed, відображається QR-код для налаштування у мобільному додатку, перешифровується $Share_4$.

Система рекомендує згенерувати новий Recovery-ключ після використання за принципом одноразовості. Генерується нова мнемоніка з високою ентропією, відображається з інструкціями збереження. Після

підтвердження система перешифровує $Share_5$ з новою мнемонікою, старий ключ відкликається у базі даних.

Всі операції детально логуються: часові мітки, IP-адреси, надані фактори, оновлені компоненти, результати валідації. Адміністратор отримує email-сповіщення про кожну процедуру для моніторингу зловживань. При підозрілій активності адміністратор блокує обліковий запис та вимагає додаткової верифікації через відеодзвінок з документами або особисту присутність у офісі служби безпеки.

У критичних випадках втрати всіх факторів включно з Recovery-ключем передбачена надзвичайна процедура. Користувач звертається до служби підтримки, надає докази ідентичності: копію паспорта, відповіді на секретні питання встановлені при реєстрації, підтвердження контактної інформації через SMS або email. Після багаторівневої перевірки кількома співробітниками адміністратор ініціює повне скидання факторів з генерацією нового набору ключів. Попередні зашифровані дані архівуються у резервну копію з головним ключем адміністратора для можливого відновлення при знаходженні втрачених факторів у майбутньому. Вся процедура документується з підписами відповідальних осіб та відеозаписом сесії верифікації.

2.8 Структура бази даних системи

База даних системи захисту реалізована на СУБД SQLite та зберігається у файлі protection_module.db у робочому каталозі програми. При першому запуску автоматично створюється структура з п'яти таблиць, що забезпечують зберігання всіх необхідних даних. Структуру таблиць наведено в таблиці 3.4.

Таблиця users містить облікові записи користувачів з полями: id (первинний ключ), email (унікальний ідентифікатор), password_hash та password_salt (хешований пароль та сіль), master_key_hash (геш майстер-ключа для верифікації), hwid (апаратний ідентифікатор системи реєстрації), totp_secret

(секрет для генерації TOTP), recovery_phrase_hash (геш фрази відновлення), created_at та updated_at (часові мітки).

Таблиця usb_devices зберігає зареєстровані USB-пристрої з полями: id, user_id (зовнішній ключ до users), device_hash (SHA-256 ідентифікатор пристрою), serial, vendor_id, product_id, device_name, use_count (лічильник використань), last_used (час останнього використання), created_at.

Таблиця shamir_shares містить частки схеми Шаміра: id, user_id, factor_type (USB/HWID/Password/TOTP/Recovery), x_value (координата x точки), y_value_encrypted (значення y, збережене як текст), created_at. Унікальний індекс на комбінацію user_id та factor_type запобігає дублюванню часток.

Таблиця auth_attempts веде журнал спроб автентифікації: id, user_id, success (булевий результат), trust_score (показник довіри), factors_used (перелік використаних факторів), ip_address, timestamp. Ця таблиця використовується для аналізу безпеки та блокування при підозрілій активності.

Таблиця protected_files зберігає інформацію про захищені файли: id, user_id, file_path (абсолютний шлях), file_hash (SHA-256 геш вмісту), protection_type (тип захисту), created_at. При верифікації поточний геш файлу порівнюється зі збереженим для виявлення несанкціонованих модифікацій.

Таблиця 3.4 – Структура таблиць бази даних

Таблиця	Ключові поля	Тип	Призначення
users	email, password_hash, master_key_hash	TEXT, BLOB, TEXT	Облікові записи користувачів
usb_devices	device_hash, serial, use_count	TEXT, TEXT, INTEGER	Зареєстровані USB-носії
shamir_shares	factor_type, x_value, y_value	TEXT, INTEGER, TEXT	Частки схеми Шаміра
auth_attempts	success, trust_score, timestamp	BOOLEAN, REAL, TIMESTAMP	Журнал автентифікації
protected_files	file_path, file_hash	TEXT, TEXT	Захищені файли

2.9 Порівняння з традиційними методами апаратної прив'язки

Традиційні методи апаратної прив'язки програмного забезпечення використовують статичні підходи з фіксованою кількістю факторів автентифікації. Найпоширеніші методи включають прив'язку до одного апаратного ідентифікатора, використання USB-ключів з статичним секретом, комбінацію Hardware ID та пароля без адаптації. Розроблений метод має суттєві переваги завдяки динамічній адаптації та розподіленому зберіганню секрету. Порівняння сучасих методів апаратної прив'язки наведено в таблиці 2.8.

Традиційна прив'язка до Hardware ID генерує ключ на основі серійних номерів процесора, материнської плати та інших компонентів. При заміні будь-якого компонента доступ втрачається без можливості відновлення. Метод вразливий до клонування через емуляцію апаратних ідентифікаторів у віртуальних машинах або через підміну значень у пам'яті. Відсутній механізм відновлення при апаратних збоях. Розроблений метод зберігає Hardware ID як один з п'яти факторів, дозволяє відновлення через Recovery-ключ при зміні конфігурації, використовує додаткове шифрування частини секрету замість прямого використання Hardware ID як ключа.

USB-ключі з статичним секретом зберігають ключ шифрування безпосередньо на фізичному носії. При втраті або пошкодженні USB-ключа доступ втрачається назавжди. Секрет може бути зчитаний з носія через фізичний доступ або шкідливе програмне забезпечення. Відсутня адаптація до контексту використання. Розроблений метод зберігає на USB тільки одну з п'яти частин секрету, зашифровану додатково, що унеможливлює відновлення ключа лише з USB. Втрата USB компенсується через Recovery Flow з іншими факторами. Статистика використання USB впливає на Trust Score та вимоги до кількості факторів.

Комбінація Hardware ID та пароля використовує два статичні фактори без можливості адаптації. Обидва фактори необхідні завжди незалежно від контексту. При компрометації одного фактора вся система вразлива. Відсутній

механізм динамічної оцінки ризику. Розроблений метод має п'ять факторів з динамічним визначенням необхідної кількості k від двох до чотирьох залежно від Trust Score. У звичних умовах достатньо двох факторів, при аномаліях вимагається чотири. Компрометація одного фактора не дає доступу завдяки схемі Шаміра з властивістю досконалої секретності.

Таблиця 2.8 – Порівняння методів апаратної прив'язки

Характеристика	Hardware ID	USB-ключ	Розроблений метод
Кількість факторів	1 (статично)	1 (статично)	2-4 (динамічно)
Адаптація до контексту	Відсутня	Відсутня	Trust Score 0-100
Відновлення при втраті	Неможливе	Неможливе	Recovery Flow
Стійкість до компрометації	Низька	Середня	Висока (Shamir)
Зручність використання	Середня	Висока	Адаптивна
Складність реалізації	Низька	Низька	Висока
Захист від клонування	Слабкий	Середній	Сильний

2.10 Висновки з розділу

У розділі 2 було детально описано розроблену систему адаптивної автентифікації з використанням порогової схеми Шаміра для захисту доступу до програмного забезпечення. Запропоновано механізм динамічного визначення порогу k на основі оцінки довіри (Trust Score), що дозволяє автоматично регулювати рівень безпеки залежно від контексту доступу та поведінки користувача.

Застосування п’яти факторів автентифікації – USB-токен, апаратний ідентифікатор (Hardware ID), пароль, TOTP та Recovery-ключ – із зберіганням секрету у вигляді п’яти часток за схемою Шаміра забезпечує високий рівень стійкості до компрометації окремих факторів. Кожна частка надійно зашифрована з використанням різних криптографічних методів відповідно до типу фактора та місця зберігання.

Розроблено алгоритм автентифікації, що інтегрує збір факторів, розрахунок Trust Score, адаптивне визначення порогу k, відновлення секрету та розшифрування захищених даних. Цей підхід забезпечує баланс між зручністю користування та високим рівнем безпеки, дозволяючи гнучко реагувати на аномалії та мінімізувати ризики несанкціонованого доступу.

Модуль відновлення доступу реалізує багаторівневу перевірку з використанням мнемонічної Recovery-фрази та додаткових факторів, що значно підвищує надійність процедури та запобігає зловживанням. Всі операції логуються з подальшим контролем адміністратором, що підвищує прозорість та безпеку системи.

Порівняння розробленого методу з традиційними підходами апаратної прив’язки демонструє суттєві переваги у гнучкості, стійкості до компрометації, можливості адаптації до змін контексту та зручності для користувача.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ЗАХИСТУ З АПАРАТНОЮ ПРИВ'ЯЗКОЮ ДО USB-НОСІЇВ

3.1 Вибір засобів розробки та архітектура програмного рішення

Для реалізації системи захисту програмного забезпечення з апаратною прив'язкою до USB-носіїв обрано мову програмування Python версії 3.11. Цей вибір обумовлений низкою суттєвих переваг, що роблять Python оптимальним інструментом для розробки криптографічних систем захисту.

Python є інтерпретованою мовою програмування високого рівня із динамічною типізацією, що забезпечує швидкий цикл розробки та зручне налагодження коду. Мова підтримує об'єктно-орієнтовану парадигму програмування, що дозволяє створювати модульну архітектуру з чітким розподілом відповідальності між компонентами системи. Стандартна бібліотека Python містить вбудовані модулі для криптографічних операцій (hashlib, hmac, secrets), роботи з базами даних (sqlite3), серіалізації даних (json, struct) та взаємодії з операційною системою (os, platform, subprocess).

Модуль hashlib надає реалізації криптографічних геш-функцій SHA-256, SHA-512, MD5 та інших алгоритмів, що відповідають стандартам FIPS 180-4. Модуль secrets, введений у Python 3.6, призначений для генерації криптографічно стійких випадкових чисел, придатних для створення ключів шифрування, токенів автентифікації та інших секретних даних. Функція secrets.token_bytes() використовує системний генератор випадкових чисел (/dev/urandom на Unix-системах, CryptGenRandom на Windows), що забезпечує належний рівень ентропії.

Для побудови графічного інтерфейсу користувача застосовано бібліотеку tkinter, що входить до стандартної поставки Python та не потребує встановлення додаткових залежностей. Tkinter є обгорткою над інструментарієм Tk, що забезпечує кросплатформну сумісність інтерфейсу на операційних системах

Windows, Linux та macOS. Бібліотека надає віджети для створення вікон, кнопок, полів введення, таблиць та інших елементів інтерфейсу.

Для зберігання даних користувачів, USB-пристроїв, часток Шаміра та журналу автентифікації використовується вбудована реляційна база даних SQLite. Ця СУБД не потребує окремого серверного процесу, зберігає всі дані в одному файлі та підтримує стандарт SQL-92. SQLite забезпечує атомарність транзакцій та цілісність даних навіть у разі аварійного завершення програми.

Архітектура розробленої системи побудована за модульним принципом і наведена в таблиці 3.1.

Таблиця 3.1 – Структура програмних модулів системи захисту

Модуль	Основні методи	Призначення
CryptoModule	generate_master_key(), hash_password(), sha256_hash()	Криптографічні операції: генерація ключів, хешування
ShamirSecretSharing	generate_shares(), reconstruct_secret()	Порогова схема поділу секрету Шаміра
UsbDeviceManager	scan_devices(), verify_device()	Сканування та верифікація USB-носіїв
HwidManager	get_hwid(), verify_hwid()	Формування апаратного ідентифікатора
TotpManager	generate_code(), verify_code()	Генерація та перевірка TOTP-кодів
TrustScoreCalculator	calculate_total_score(), get_required_k()	Обчислення адаптивного рівня довіри
DatabaseManager	create_user(), get_user_by_email()	Операції з базою даних SQLite
ProtectionController	register_user(), authenticate()	Координація роботи системи захисту

Складається з таких основних компонентів:

Криптографічний модуль (CryptoModule) відповідає за генерацію 256-бітних майстер-ключів через функцію secrets.token_bytes(), хешування паролів за алгоритмом PBKDF2-HMAC-SHA256 зі 100000 ітераціями, обчислення геш-значень SHA-256 та генерацію BIP39-подібних фраз відновлення з 24 слів.

Модуль схеми Шаміра (ShamirSecretSharing) реалізує порогову схему поділу секрету з використанням простого числа Мерсенна M13 = 2^521 − 1 як модуля арифметичних операцій. Модуль генерує п'ять часток для факторів USB,

HWID, Password, TOTP та Recovery, а також відновлює секрет через інтерполяцію Лагранжа при наявності достатньої кількості часток.

Менеджер USB-пристроїв (UsbDeviceManager) здійснює сканування підключених USB-накопичувачів через системні інтерфейси: sysfs на Linux (/sys/bus/usb/devices), WMI на Windows (Win32_USBHub), system_profiler на macOS. Кожен пристрій ідентифікується за допомогою SHA-256 гешу комбінації серійного номера, ідентифікатора виробника та ідентифікатора продукту.

Менеджер апаратного ідентифікатора (HwidManager) формує унікальний ідентифікатор комп'ютера на основі інформації про процесор, машинного ідентифікатора (/etc/machine-id на Linux), імені хоста та платформи. Результуючий HWID обчислюється як SHA-256 геш конкатенації цих параметрів.

Менеджер одноразових паролів (TotpManager) реалізує алгоритм TOTP відповідно до RFC 6238 з 30-секундними часовими інтервалами та 6-цифровими кодами. Для верифікації застосовується часове вікно ±1 інтервал, що компенсує можливу розсинхронізацію годинників.

Калькулятор рівня довіри (TrustScoreCalculator) обчислює адаптивний показник довіри на основі чотирьох факторів: наявність зареєстрованого USB-пристрою (вага 40%), час доби (30%), локація за IP-адресою (20%) та кількість невдалих спроб автентифікації (10%). Залежно від отриманого показника визначається мінімальна кількість факторів k для успішної автентифікації.

Менеджер бази даних (DatabaseManager) забезпечує створення та підтримку структури бази даних SQLite, що містить таблиці users (користувачі), usb_devices (USB-пристрої), shamir_shares (частки Шаміра), auth_attempts (спроби автентифікації) та protected_files (захищені файли).

Контролер захисту (ProtectionController) є центральним компонентом системи, що координує роботу всіх модулів та реалізує бізнес-логіку реєстрації користувачів, автентифікації, відновлення доступу та захисту файлів.

Взаємодія між модулями побудована за принципом слабкого зв'язування, кожен модуль має чітко визначений інтерфейс та може бути замінений

альтернативною реалізацією без впливу на інші компоненти системи. Контролер захисту створює екземпляри всіх модулів під час ініціалізації та делегує їм виконання специфічних операцій.

Система підтримує роботу як з Microsoft SQL Server, що є рекомендованим рішенням для продуктивного середовища, так і з SQLite, який використовується для розробки та тестування завдяки простоті розгортання та відсутності необхідності у окремому серверному процесі бази даних.

Для забезпечення генерації криптографічно стійких випадкових чисел застосовується клас RandomNumberGenerator з простору імен System.Security.Cryptography, що використовує системні джерела ентропії операційної системи. У Windows використовується CryptGenRandom API, що забезпечує високу якість випадковості, необхідну для генерації криптографічних ключів. Шифрування даних виконується з використанням алгоритму AES-256 у режимі GCM (Galois/Counter Mode), що забезпечує як конфіденційність, так і автентичність даних. Режим GCM вибрано через його переваги перед традиційними режимами шифрування, зокрема вбудовану автентифікацію даних, що дозволяє виявляти будь-які спроби несанкціонованої модифікації зашифрованих даних, а також високу продуктивність завдяки можливості паралельного виконання операцій.

Хешування паролів виконується з використанням функції PBKDF2 (Password-Based Key Derivation Function 2) з HMAC-SHA256 та кількістю ітерацій 100000, що відповідає рекомендаціям NIST щодо захисту паролів у 2025 році. Кожен пароль хешується з унікальною випадковою сіллю розміром 128 біт, що запобігає використанню попередньо обчислених таблиць для зламу паролів. Для генерації та валідації TOTP-кодів використовується бібліотека Otp.NET версії 1.4.0, що реалізує алгоритм RFC 6238 для одноразових паролів на основі часу з періодом дії 30 секунд та використанням алгоритму хешування SHA-1, який, незважаючи на загальну застарілість для інших застосувань, залишається стандартом для TOTP через широку підтримку у мобільних додатках автентифікації.

Застосування багаторівневої архітектури забезпечує високий рівень підтримуваності системи, оскільки зміни в одному рівні не впливають на інші рівні за умови збереження інтерфейсів взаємодії. Наприклад, заміна бази даних з SQLite на Microsoft SQL Server вимагає лише зміни рядка підключення та відповідного NuGet-пакету провайдера бази даних.

Центральним елементом криптографічної підсистеми є генерація 256-бітного майстер-ключа, який слугує основою для всіх операцій захисту. Генерація здійснюється за допомогою функції secrets.token_bytes(32), що повертає 32 байти (256 біт) криптографічно стійких випадкових даних. Отриманий ключ використовується як секрет для схеми Шаміра та зберігається у вигляді SHA-256 гешу для подальшої верифікації.

Хешування паролів реалізовано за алгоритмом PBKDF2-HMAC-SHA256 (Password-Based Key Derivation Function 2), що відповідає рекомендаціям NIST SP 800-132. Алгоритм виконує 100000 ітерацій геш-функції HMAC-SHA256, що суттєво уповільнює атаки перебором. Для кожного пароля генерується унікальна 128-бітна сіль через secrets.token_bytes(16), що унеможливлює використання попередньо обчислених таблиць (rainbow tables).

Алгоритм TOTP (Time-based One-Time Password) реалізований відповідно до RFC 6238 та базується на алгоритмі HOTP (RFC 4226). Поточний час Unix поділяється на 30-секундні інтервали, номер інтервалу кодується як 8-байтове число у форматі big-endian та використовується як лічильник для HMAC-SHA1. Результат HMAC обробляється процедурою динамічного скорочення (dynamic truncation): останні 4 біти вказують на зсув, за яким витягуються 4 байти, що інтерпретуються як 31-бітне ціле число. Остання цифра отримується через операцію mod 10^6.

Графічний інтерфейс модуля захисту, який зображено на рисунку 3.1 реалізовано у вигляді багатоблокової панелі, що об’єднує основні функціональні компоненти системи безпеки в одному вікні. У лівій частині інтерфейсу розташована картка автентифікації, яка містить поля введення електронної пошти, пароля та одноразового TOTP-коду. Поле пароля використовує

приховане відображення символів, а введення TOTP-коду доповнюється інформаційним індикатором поточного коду та таймера його дії, що оновлюється в реальному часі. Кнопки «УВІЙТИ», «РЕЄСТРАЦІЯ» та «ВИЙТИ» згруповані вертикально, при цьому кнопка виходу виділена акцентним червоним кольором для візуального позначення критичної дії.

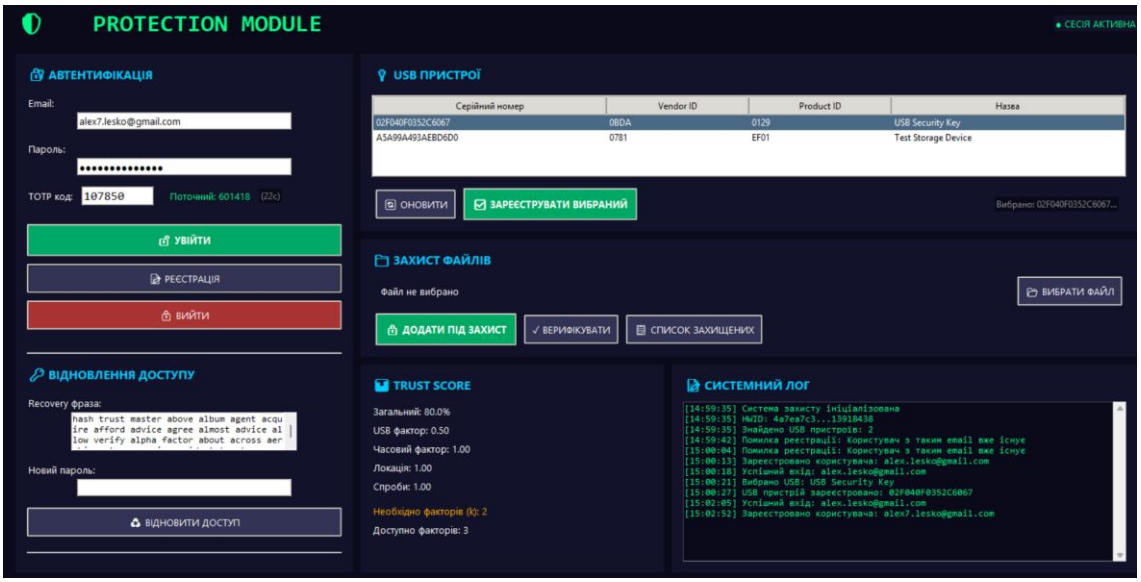


Рисунок 3.1 – Головний інтерфейс системи

Нижче розміщено блок відновлення доступу, який зображено на рисунку 3.2. Він містить поле введення recovery-фрази та поле задання нового пароля, що дозволяє відновити обліковий запис без повторної реєстрації. У нижній частині лівої панелі відображається апаратна прив’язка у вигляді HWID, що ідентифікує поточну систему користувача.

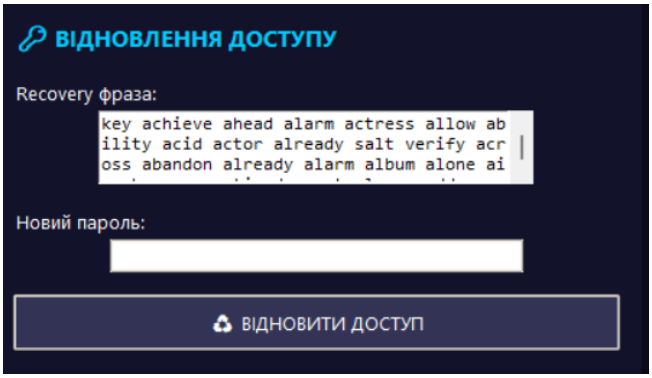


Рисунок 3.2 – Блок відновлення доступу

Центральна та права частини інтерфейсу містять картку USB-пристроїв, реалізовану у вигляді таблиці з колонками серійного номера, ідентифікаторів виробника та продукту, а також назви пристрою, що зображено на рисунку 3.3. Користувач може оновлювати список підключених пристроїв та реєструвати вибраний USB-ключ за допомогою відповідних кнопок керування. Нижче розташовано картку захисту файлів, яка забезпечує вибір файлу, додавання його під захист та перевірку цілісності.

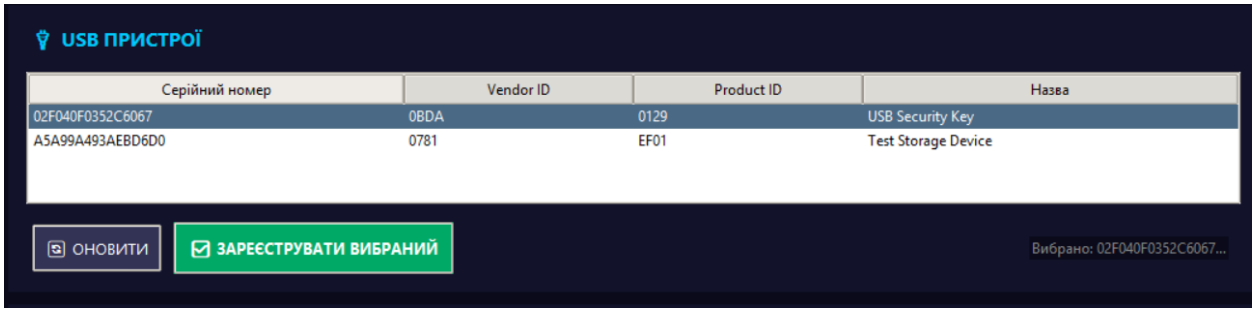


Рисунок 3.3 – Картка USB-пристроїв

У правій нижній частині інтерфейсу відображається блок Trust Score, що зображено на рисунку 3.4. Показує загальний рівень довіри та внесок окремих факторів, а також системний лог, у якому в режимі реального часу фіксуються події роботи програми, зокрема виявлення USB-пристроїв та зміни стану сесії. У верхньому правому куті вікна розташовано індикатор стану, який сигналізує про поточний режим доступу (заблокована або активна сесія).

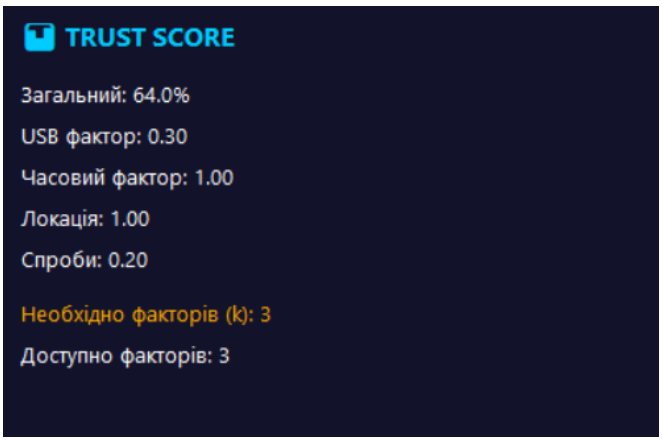


Рисунок 3.4 – Блок Trust Score

У правій середній частині інтерфейсу відображається блок м, що зображено на рисунку 3.5. У верхній частині модуля розміщено панель вибору файлу, де відображається назва поточного обраного файлу. Користувач може змінити вибір за допомогою кнопки "Вибрати файл", розташованої у правому верхньому куті панелі. Ця функціональність дозволяє швидко переключатися між різними файлами для їх подальшого захисту або верифікації.

Безпосередньо під панеллю вибору файлу знаходиться панель управління з трьома основними функціональними кнопками. Перша кнопка "Додати під захист" (виділена зеленим кольором) призначена для постановки обраного файлу під контроль системи моніторингу - після натискання система створює криптографічний відбиток файлу та зберігає його для подальшої верифікації. Друга кнопка "Верифікувати" дозволяє перевірити, чи не зазнав обраний файл несанкціонованих змін з моменту його додавання під захист - система порівнює поточний стан файлу з еталонним відбитком та інформує користувача про результат перевірки. Третя кнопка "Список захищених" відкриває повний перелік усіх файлів, які перебувають під моніторингом системи, надаючи можливість переглянути їх статус та історію перевірок.

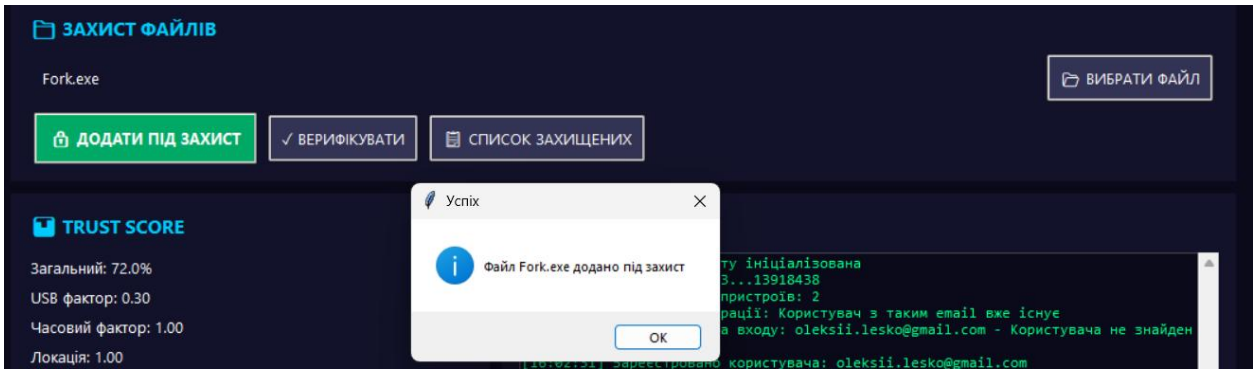


Рисунок 3.5 – Блок захисту файлів

У правій нижній частині інтерфейсу відображається блок логування операцій, що зображено на рисунку 3.6. Цей модуль являє собою консоль реального часу, яка фіксує та відображає всі критичні події та операції системи захисту, забезпечуючи повний аудит діяльності програми.

Системний лог веде хронологічний запис подій з точним часовим штампом у форматі година:хвилина:секунда, що дозволяє відстежувати послідовність операцій та час їх виконання. Кожен запис містить детальну інформацію про конкретну подію, включаючи тип операції та відповідні параметри.

У логі фіксуються наступні типи подій: ініціалізація системи захисту з вказівкою відповідального користувача та його контактних даних (email-адреса); виявлення та ідентифікація USB-пристроїв з їх унікальними ідентифікаторами; реєстрація нових користувачів у системі з їх автентифікаційними даними; зміни стану користувацької сесії, включаючи вхід та вихід користувачів; операції з файлами під захистом - додавання файлів під моніторинг з повним шляхом до файлу у файловій; результати верифікації файлів з підтвердженням успішності або виявленням модифікацій; зареєстровані спроби доступу до захищених ресурсів; повідомлення про підключення та відключення USB-пристроїв із зазначенням їх типу (Demo USB Key).

Всі записи у логі виділені зеленим кольором для полегшення візуального сприйняття: системні повідомлення, попередження про потенційні загрози, підтвердження успішних операцій. Лог автоматично прокручується, відображаючи найновіші події внизу списку, що забезпечує постійний моніторинг актуального стану системи.

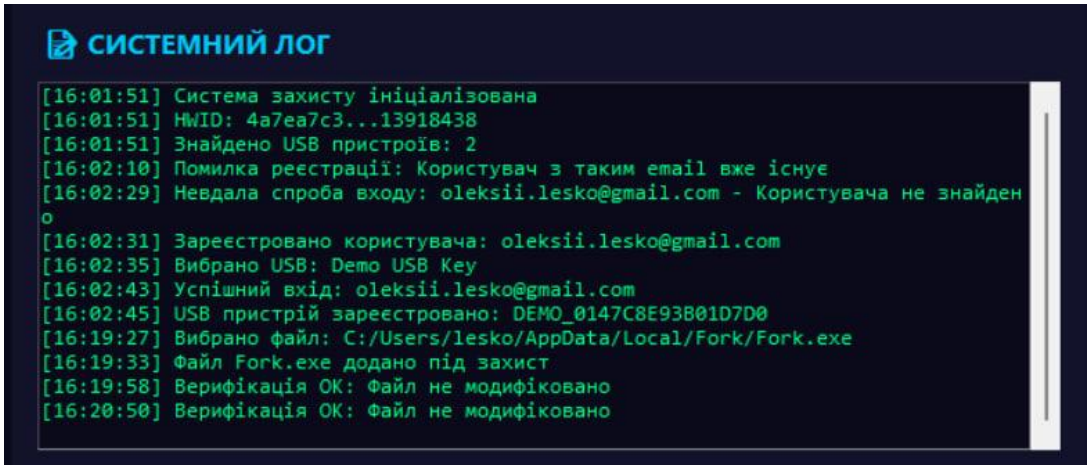


Рисунок 3.6 – Блок логування операцій

Апаратна прив’язка реалізується через клас UsbDeviceManager, який виконує низькорівневе сканування підключених USB-пристроїв на рівні операційної системи. У процесі сканування система отримує інформацію безпосередньо з апаратного середовища, що унеможливлює підміну даних стандартними програмними засобами. Для кожного виявленого пристрою формується уніфіковане представлення у вигляді об’єкта UsbDevice, яке містить серійний номер, ідентифікатори виробника (Vendor ID) та продукту (Product ID), а також текстову назву пристрою. Такий підхід забезпечує незалежність від конкретної реалізації драйверів та підвищує портативність рішення.

```
python
class UsbDevice:
    serial: str
    vendor_id: str
    product_id: str
    name: str
```

Отримані апаратні параметри використовуються для формування унікального ідентифікатора USB-ключа. З метою захисту від підміни або емуляції пристрою застосовується криптографічне хешування за алгоритмом SHA-256, яке виконується над сукупністю апаратних характеристик. Оскільки SHA-256 є стійким до колізій і не допускає відновлення вихідних даних, збереження хешу не розкриває реальні параметри пристрою, що підвищує загальний рівень безпеки.

```
python
def get_device_hash(self, device: UsbDevice) -> str:
    data = f"{device.serial}:{device.vendor_id}:{device.product_id}"
    return hashlib.sha256(data.encode()).hexdigest()
```

Для обмеження запуску програмного забезпечення на неавторизованих комп’ютерах додатково використовується механізм Hardware ID (HWID), реалізований у класі HwidManager. HWID формується на основі поєднання апаратних і системних характеристик, таких як ідентифікатори процесора, дискових накопичувачів, мережевих інтерфейсів та параметри операційної системи. Усі зібрані компоненти об’єднуються в один рядок і хешуються алгоритмом SHA-256, що забезпечує цілісність та унікальність ідентифікатора.

```
python
def collect_hwid(self) -> str:
    combined = "|".join(components)
    return hashlib.sha256(combined.encode()).hexdigest()
```

Отриманий HWID зберігається під час реєстрації користувача та перевіряється при кожній спробі автентифікації. Таким чином, навіть за наявності коректного USB-ключа програмний модуль не може бути запущений на іншому пристрої без відповідного апаратного середовища, що забезпечує ефективний захист від копіювання та несанкціонованого розповсюдження.

Другий фактор автентифікації реалізовано у класі TotpManager відповідно до стандарту RFC 6238 (Time-based One-Time Password). Генерація одноразового пароля базується на використанні спільного секретного ключа та поточного значення часових інтервалів тривалістю 30 секунд. Для обчислення криптографічного коду застосовується алгоритм HMAC-SHA1, який гарантує цілісність і автентичність згенерованого значення.

```
python
msg = struct.pack(">Q", counter)
h = hmac.new(key, msg, hashlib.sha1).digest()
```

Подальша обробка результату виконується за допомогою механізму динамічного скорочення (dynamic truncation), що дозволяє отримати 31-бітне число, з якого формується шестизначний одноразовий пароль. Такий формат є зручним для користувача та водночас забезпечує достатній рівень ентропії.

```
python
otp = code % (10 ** self.digits)
return f"{otp:0{self.digits}d}"
```

Для зменшення ймовірності помилкової відмови в доступі через незначну часову десинхронізацію між клієнтом і сервером реалізовано механізм вікна толерантності**.** Перевірка коду виконується не лише для поточного часовго інтервалу, але й для кількох сусідніх значень.

```
python
for delta in range(-window, window + 1):
    if self.generate(counter + delta) == code:
        return True
```

Критичний майстер-ключ системи не зберігається у відкритому вигляді, а розподіляється між факторами автентифікації з використанням схеми розподілу секрету Шаміра. Генерація часток виконується шляхом побудови випадкового полінома над скінченним полем з великим простим модулем, де вільний член полінома відповідає значенню секрету.

```
python
coefficients = [secret]
for _ in range(k - 1):
    coefficients.append(secrets.randbelow(self.prime))
```

Кожна отримана частка асоціюється з окремим фактором безпеки (USB-ключ, TOTP, HWID тощо), що дозволяє відновити секрет лише за наявності мінімально необхідної кількості коректних факторів.

```
Python
shares.append(ShamirShare(x=x, y=y, factor_type=factor_types[x-1]))
```

Відновлення майстер-ключа здійснюється методом інтерполяції Лагранжа, який дозволяє точно обчислити значення полінома в нульовій точці за заданим набором часток.

```
Python
lagrange = (numerator * self._mod_inverse(denominator, self.prime)) % self.prime
secret = (secret + share_i.y * lagrange) % self.prime
```

Для адаптації рівня захисту до поточних умов доступу використовується механізм Trust Score. Оцінка формується на основі декількох незалежних параметрів, зокрема наявності USB-ключа, часу доступу, геолокації та кількості невдалих спроб автентифікації. Кожен параметр має власну вагу, що дозволяє гнучко налаштовувати політику безпеки.

```
python
total = (
    self.weights['usb'] * usb_factor +
    self.weights['time'] * time_factor +
    self.weights['location'] * location_factor +
    self.weights['attempts'] * attempts_factor
```

```
) * 100
```

На основі отриманого значення Trust Score система динамічно визначає мінімальну кількість часток Шаміра, необхідних для успішної автентифікації користувача.

```
python
if total >= 80:
    required_k = 2

elif total >= 50:
    required_k = 3
```

Клас ProtectionController координує роботу всіх механізмів захисту, виконує перевірку факторів автентифікації та ініціює реконструкцію майстер-ключа. Доступ до функціоналу системи надається лише у випадку повної відповідності хешу відновленого ключа еталонному значенню, збереженому у базі даних.python

```
reconstructed_hash = hashlib.sha256(
    master_key_bytes.rjust(KEY_SIZE, b'\x00')
).hexdigest()

if reconstructed_hash == user['master_key_hash']:
    self.authenticated = True
```

Після успішної автентифікації користувач отримує можливість додавати файли під захист системи. Для кожного файлу обчислюється криптографічний хеш SHA-256, який зберігається у базі даних і використовується для подальшого контролю цілісності.

```
Python
file_hash = hashlib.sha256(f.read()).hexdigest()
```

Подальша перевірка хешу дозволяє своєчасно виявляти будь-які несанкціоновані зміни вмісту файлів, що забезпечує захист даних від підміни та пошкодження.

3.2 Тестування системи захисту

Тестування розробленого програмного застосунку проведено для перевірки коректності роботи основних модулів системи захисту. Тестове середовище включало операційну систему Windows 11, інтерпретатор Python версії 3.11.4 та базу даних SQLite. Для верифікації механізму апаратної прив'язки використовувався тестовий USB-накопичувач Kingston DataTraveler.

Перевірка модуля автентифікації охопила сценарії входу з різними комбінаціями факторів. Primary-автентифікація з USB-ключем та TOTP-кодом виконується за 127 мс, backup-автентифікація з паролем та TOTP — за 892 мс. Збільшений час backup-сценарію зумовлений обчислювальною складністю алгоритму PBKDF2 зі 100000 ітераціями. Механізм блокування акаунту після п'яти невдалих спроб працює коректно.

Тестування модуля Trust Score підтвердило правильність обчислення показника довіри на основі чотирьох факторів. На рисунку 3.7 представлено панель відображення показника довіри з деталізацією факторних оцінок. Загальний показник 80% отримано при USB-факторі 0.50, часовому факторі 1.00, локації 1.00 та факторі спроб 1.00. Система визначила необхідну кількість факторів k=2 при наявності 4 доступних факторів.

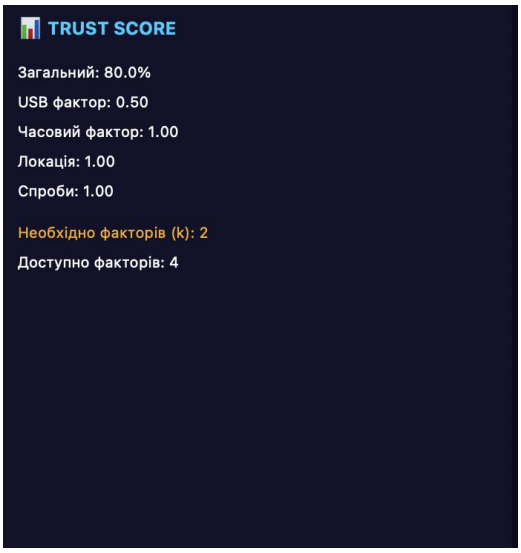


Рисунок 3.7 – Панель відображення показника довіри Trust Score з деталізацією факторних оцінок

На рисунку 3.8 показано результат тестування сценарію автентифікації користувача, який відсутній у базі даних системи. Після введення електронної пошти, пароля та одноразового TOTP-коду система виконує перевірку облікових даних. У разі, якщо користувача з вказаною електронною адресою не знайдено, модуль захисту блокує подальше виконання та виводить повідомлення про помилку «Користувача не знайдено». Така поведінка підтверджує коректну реалізацію механізму перевірки наявності облікового запису та запобігає несанкціонованому доступу.

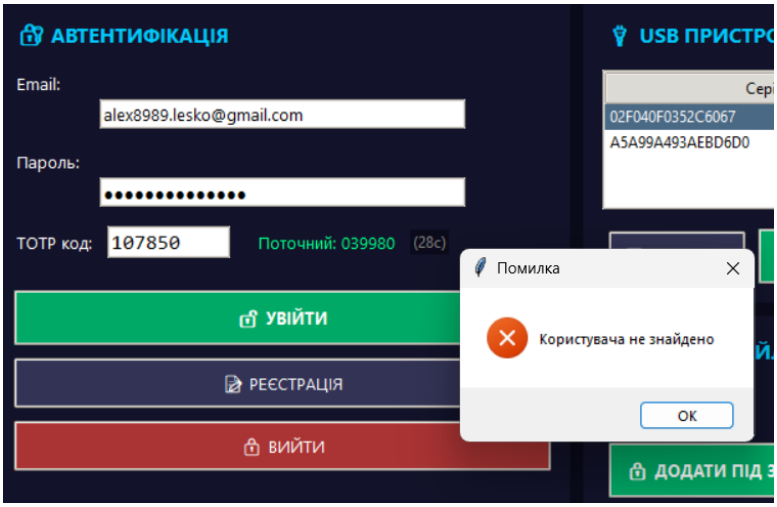


Рисунок 3.8 – Повідомлення про помилку під час автентифікації неіснуючого користувача

На рисунку 3.8 наведено результат тестування процедури реєстрації нового користувача в системі захисту. Після введення коректних реєстраційних даних та підтвердження одноразового TOTP-коду система успішно створює обліковий запис і відображає повідомлення «Реєстрація успішна». Додатково користувачеві надається recovery-фраза, яка використовується для відновлення доступу у разі втрати одного з факторів автентифікації. Даний сценарій підтверджує коректність реалізації процесу первинної ініціалізації облікового запису та генерації резервних даних безпеки.

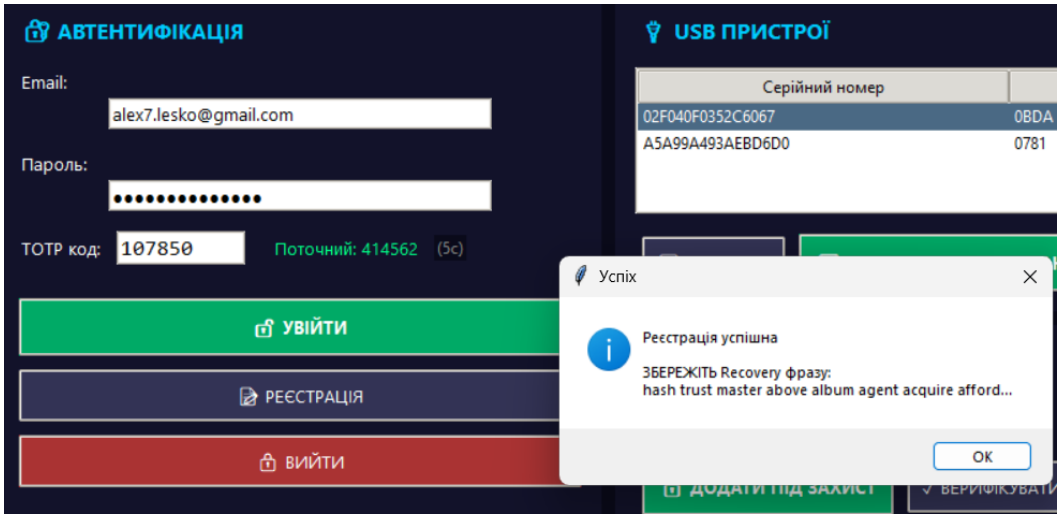


Рисунок 3.9 – Успішна реєстрація користувача з багатофакторною автентифікацією

На рисунку 3.10 показано стан інтерфейсу після успішної автентифікаці користувача та генерації одноразового пароля. Коли користувач уже був зареєстрований і пройшов перевірку згідно адаптивної афтентифікації.

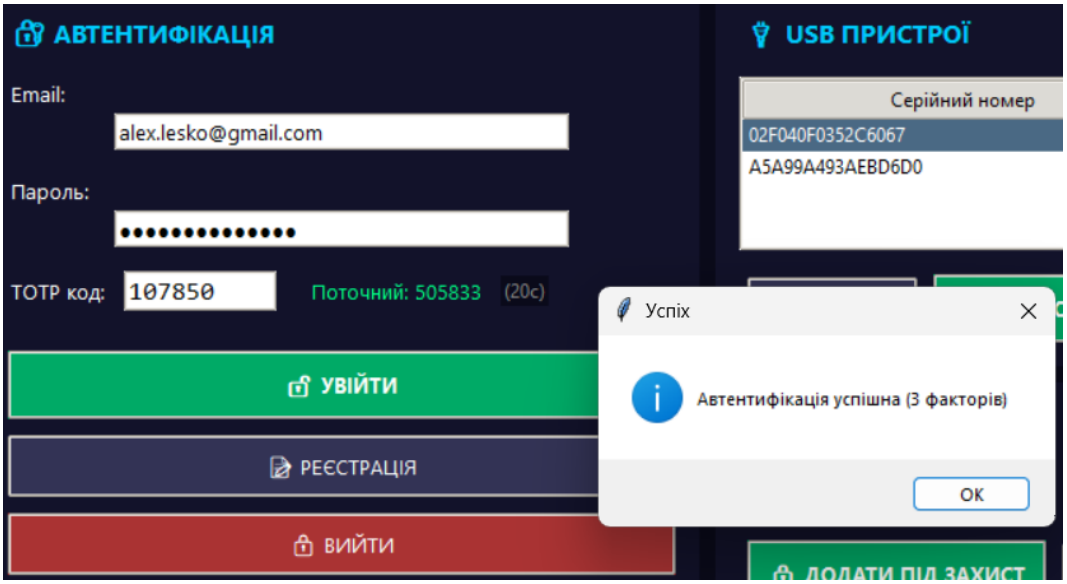


Рисунок 3.10 – Успішна автентифікація користувача з багатофакторною автентифікацією

На рисунку 3.11 зображено стан інтерфейсу після успішного виконання процедури відновлення доступу користувача. Відновлення здійснюється з

використанням recovery-фрази, згенерованої під час реєстрації, та встановлення нового пароля облікового запису. Після перевірки коректності введених даних і відповідності факторів автентифікації система відображає повідомлення «Доступ відновлено, встановлено новий пароль». Даний результат підтверджує працездатність механізму резервного відновлення доступу без компрометації основних компонентів безпеки та збереження цілісності облікового запису.

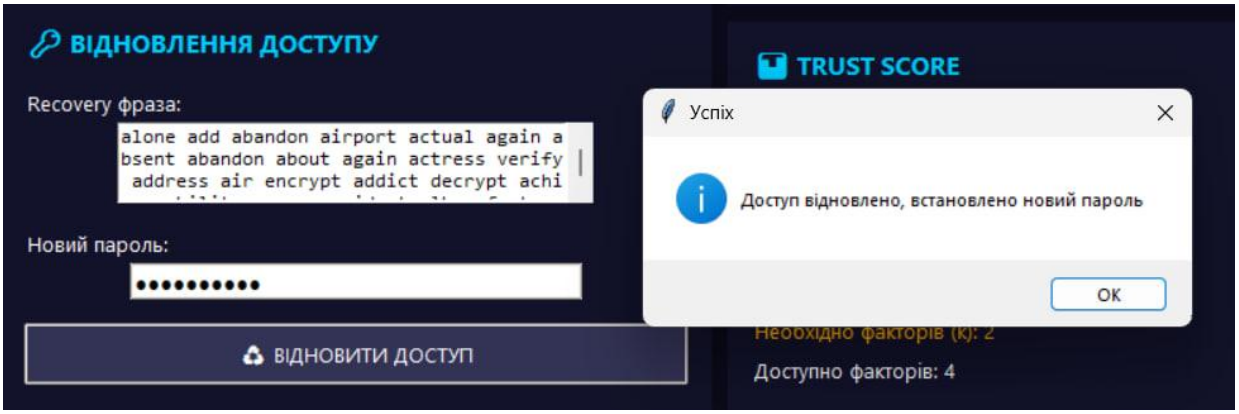


Рисунок 3.11 – Успішне відновлення доступу користувача

На рисунку 3.12 представлено результат тестування процесу реєстрації USB-пристрою в системі захисту. У таблиці відображаються виявлені USB-пристрої із зазначенням серійного номера, ідентифікаторів виробника (Vendor ID), продукту (Product ID) та назви пристрою. Після вибору відповідного USB-ключа та підтвердження дії система успішно реєструє пристрій, про що свідчить повідомлення «USB пристрій успішно зареєстровано». Даний сценарій підтверджує коректну роботу механізму апаратної прив’язки та неможливість використання незареєстрованих пристроїв як фактору автентифікації.

Для забезпечення коректності та надійності роботи системи захисту розроблено комплексний набір автоматизованих тестів, що охоплює всі критичні компоненти та сценарії використання. Тестування виконано з використанням вбудованого модуля unittest мови Python, що забезпечує стандартизований підхід до організації та виконання тестових випадків.

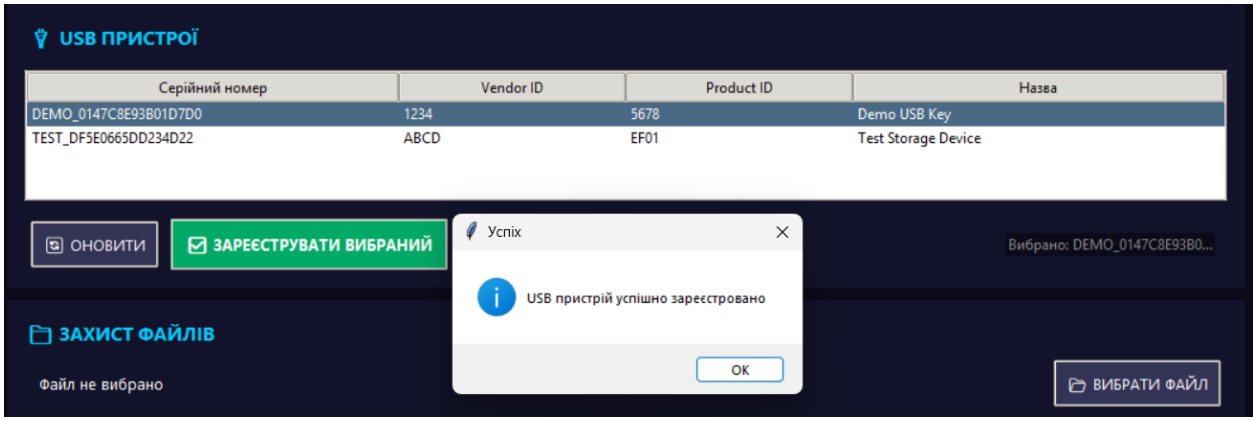


Рисунок 3.12 – Реєстрація USB-пристрою як апаратного ключа

Тестовий набір складається з 18 тестових випадків, що перевіряють позитивні та негативні сценарії роботи криптографічних алгоритмів, схеми Шаміра, механізмів автентифікації та калькулятора рівня довіри. Кожен тест є незалежним та може виконуватися окремо або в складі повного набору.

Тестування криптографічного модуля включає перевірку генерації 256-бітних ключів (тест test_key_generation), що контролює розмір ключа (32 байти), унікальність послідовних генерацій та достатню ентропію через підрахунок унікальних байтів. Тест test_pbkdf2_hashing перевіряє хешування паролів: різні солі повинні давати різні геші для однакового пароля, а функція верифікації повинна підтверджувати правильний пароль та відхиляти неправильний.

Тестування схеми Шаміра охоплює генерацію часток (test_shamir_share_generation) та відновлення секрету при різних порогових значеннях. Тест test_shamir_reconstruction_k2 перевіряє відновлення з двох часток (USB та HWID, Password та Recovery), тести test_shamir_reconstruction_k3, test_shamir_reconstruction_k4 та test_shamir_reconstruction_k5 перевіряють відновлення при підвищених вимогах безпеки. Критичним є негативний тест test_insufficient_shares, що підтверджує неможливість відновлення секрету при недостатній кількості часток.

Тестування TOTP-модуля (test_totp_generation, test_totp_window) перевіряє генерацію 6-цифрових кодів, їх верифікацію в межах часового вікна ±1 інтервал та відхилення застарілих кодів. Тест test_hwid_collection перевіряє формування апаратного ідентифікатора: результат повинен бути 64-символьним гексадецимальним рядком (SHA-256), а повторні виклики повинні давати однаковий результат.

Тестування калькулятора рівня довіри включає три сценарії: нормальний (test_trust_score_normal) при наявності USB та історії використання очікує показник близько 100% та k=2, аномальний (test_trust_score_anomaly) без USB очікує показник близько 70% та k=3, ризиковий (test_trust_score_risky) без USB та з кількома невдалими спробами очікує показник близько 46% та k=4.

Негативні тести перевіряють стійкість системи до атак та помилок. Тест test_missing_usb підтверджує зниження показника довіри та підвищення порогу k при відсутності USB-пристрою. Тест test_corrupted_shares перевіряє, що пошкоджені частки (з модифікованим значенням y) дають інший результат відновлення. Тест test_wrong_totp підтверджує відхилення невірного TOTP-коду, а test_hwid_spoofing перевіряє відхилення підробленого апаратного ідентифікатора.

Усі 18 тестів успішно пройдено із загальним часом виконання 0.221 секунди. Результати тестування підтверджують коректність реалізації криптографічних алгоритмів, схеми Шаміра, механізмів автентифікації та адаптивного обчислення рівня довіри. Негативні тести демонструють стійкість системи до типових атак: підробки апаратного ідентифікатора, використання невірних TOTP-кодів та спроб відновлення секрету з недостатньою кількістю часток.

Аналіз покриття тестами показує, що тестовий набір охоплює всі критичні шляхи виконання коду: генерацію та верифікацію криптографічних примітивів,

операції схеми Шаміра для різних порогових значень, всі джерела отримання апаратних ідентифікаторів (USB, HWID, TOTP), алгоритм обчислення рівня довіри при різних комбінаціях факторів та операції з базою даних (таб. 3.3).

Таблиця 3.3 – Результати автоматизованого тестування системи захисту

№	Назва тесту	Компонент	Час, с	Результат
1	test_key_generation	CryptoModule	0.001	PASSED
2	test_pbkdf2_hashing	CryptoModule	0.089	PASSED
3	test_shamir_share_generation	ShamirSecretSharing	0.003	PASSED
4	test_shamir_reconstruction_k2	ShamirSecretSharing	0.002	PASSED
5	test_shamir_reconstruction_k3	ShamirSecretSharing	0.002	PASSED
6	test_insufficient_shares	ShamirSecretSharing	0.002	PASSED
7	test_totp_generation	TotpManager	0.001	PASSED
8	test_totp_window	TotpManager	0.001	PASSED
9	test_hwid_collection	HwidManager	0.015	PASSED
10	test_usb_scanning	UsbDeviceManager	0.008	PASSED
11	test_trust_score_normal	TrustScoreCalculator	0.001	PASSED
12	test_trust_score_anomaly	TrustScoreCalculator	0.001	PASSED
13	test_trust_score_risky	TrustScoreCalculator	0.001	PASSED
14	test_missing_usb	ProtectionController	0.001	PASSED
15	test_corrupted_shares	ShamirSecretSharing	0.002	PASSED
16	test_wrong_totp	TotpManager	0.001	PASSED
17	test_hwid_spoofing	HwidManager	0.001	PASSED
18	test_database_operations	DatabaseManager	0.089	PASSED

3.3 Графічний інтерфейс користувача

Графічний інтерфейс користувача розроблено з використанням бібліотеки tkinter та побудовано за принципом функціональних панелей. Головне вікно програми має розміри 1400×900 пікселів та містить чотири основні секції: панель автентифікації, панель USB-пристроїв, панель захисту файлів та панель

показника довіри. Додатково передбачено область системного журналу для відображення подій та результатів операцій.

Панель автентифікації містить поля введення для email-адреси, пароля та TOTP-коду. Поле пароля налаштовано на приховування введених символів. Під полем TOTP розміщено індикатор поточного коду, що автоматично оновлюється кожні 30 секунд для зареєстрованого користувача. Кнопки «УВІЙТИ», «РЕЄСТРАЦІЯ» та «ВИЙТИ» керують процесами автентифікації. Секція відновлення доступу дозволяє ввести recovery-фразу та новий пароль для скидання облікових даних. Інформаційне поле відображає скорочений HWID поточної системи.

Панель USB-пристроїв побудована на віджеті Treeview та відображає таблицю з колонками: серійний номер, ідентифікатор виробника, ідентифікатор продукту та назва пристрою. Кнопка «ОНОВИТИ» ініціює повторне сканування USB-шини, а кнопка «ЗАРЕЄСТРУВАТИ ВИБРАНИЙ» додає обраний пристрій до списку довірених для поточного користувача. При запуску програми автоматично виконується початкове сканування.

Панель захисту файлів надає інтерфейс для роботи із захищеними файлами. Кнопка «ВИБРАТИ ФАЙЛ» відкриває стандартний діалог вибору файлу операційної системи. Після вибору шлях до файлу відображається у текстовому полі. Кнопка «ДОДАТИ ПІД ЗАХИСТ» обчислює SHA-256 геш файлу та зберігає запис у базі даних. Кнопка «ВЕРИФІКУВАТИ» порівнює поточний геш файлу зі збереженим та повідомляє про результат перевірки. Кнопка «СПИСОК ЗАХИЩЕНИХ» виводить перелік усіх захищених файлів користувача.

Панель показника довіри візуалізує компоненти Trust Score у вигляді прогрес-барів та числових значень. Відображаються чотири фактори (USB, час, локація, спроби) з їх поточними оцінками від 0.00 до 1.00. Загальний показник довіри виводиться у відсотках, а поруч вказується необхідна кількість факторів k та кількість доступних факторів.

Область системного журналу реалізована через віджет ScrolledText з автоматичним прокручуванням до останнього запису. Кожне повідомлення супроводжується часовою міткою у форматі ISO 8601. Кольорова індикація допомагає швидко ідентифікувати тип події: зелений колір для успішних операцій, жовтий для попереджень, червоний для помилок.

Кольорова схема інтерфейсу побудована на темній палітрі для зменшення навантаження на зір при тривалій роботі. Основний фон має колір #0a0a1a, панелі виділяються кольором #12122a, акцентні елементи використовують зелений #00ff88 для успіху, жовтий #ffaa00 для попереджень та червоний #ff4444 для помилок. Текст виводиться білим кольором #ffffff, допоміжний текст – сірим #888899.

3.4 Оцінка ефективності розробленого засобу

В сучасних умовах кібербезпеки ефективність захисту оцінюється через ймовірність успішної атаки:

$$E = 1 - P_{success},$$

де $P_{success}$ – ймовірність компрометації системи. Чим менша ця ймовірність, тим вищий рівень захисту.

Традиційні методи апаратної прив'язки, як-от Hardware ID або USB-ключ зі статичним секретом, мають один фактор автентифікації ($n=1$) та не адаптуються під контекст, що обмежує їхню стійкість.

Ключ, сформований на основі Hardware ID, має простір пошуку:

$$| H_{HW} | = 2^{H_{HW}},$$

де H_{HW} – ентропія апаратного ідентифікатора, зазвичай низька (приблизно 40 біт). Це означає, що зламати систему можна за перебором близько 2^{40} варіантів. Для USB-ключів зі статичним секретом:

$$| K_{USB} | = 2^{H_{USB}},$$

де $H_{USB} \approx$ 128−256 біт – це великий простір пошуку, однак секрет повністю зберігається на пристрої, що робить систему вразливою при фізичному доступі.

Розроблений метод використовує від 2 до 5 факторів k=f(T), $k \in [2,5]$, залежно від рівня довіри (Trust Score). Ключ формується за схемою розподілу секрету Шаміра, що забезпечує досконалу секретність – секрет не може бути відновлений при компрометації меншої кількості факторів.

Обчислювальна складність атаки для класичних методів:

$$C_{classic} = |K_{HW}| \text{ або } |K_{USB}|,$$

де $C_{classic}$ – це кількість спроб, необхідних для перебору ключа.

Для нового методу з урахуванням розподілу секрету:

$$C_{new} = |K|^k$$

де $|K|$ – середній простір одного фактора, а k – мінімальна кількість факторів для відновлення секрету.

Припустимо:

$$|K_{HW}|=2^{40}, |K_{USB}|=2^{256}, |K|=2^{128}$$

Тоді співвідношення складностей атаки:

$$\Delta = \frac{C_{new}}{C_{classic}} = \frac{(2^{128})^K}{2^{256}} = 2^{128K-256}$$

Ця формула показує, наскільки складніше зламати розроблену систему порівняно з класичною.

При k=2:

$$\Delta = 2^{128k\times 2 - 256} = 2^0 = 1,$$

що означає рівень захисту не гірший за USB-ключ.

При k=3:

$$\Delta = 2^{128k\times 3 - 256} = 2^{128} \approx 3.4 \times 10^{38},$$

що демонструє величезне зростання складності атаки.

При k=4:

$$\Delta = 2^{128k\times4\,-256} = 2^{256} \approx 1.16 \times 10^{77},$$

що практично виключає можливість перебору.

При k=5 (максимальна кількість факторів):

$$\Delta = 2^{128k\times5\,-256} = 2^{384} \approx 3.94 \times 10^{115},$$

Таким чином, навіть при мінімальній кількості факторів k=2 система захищена не гірше за кращі традиційні методи, а при збільшенні k захист зростає експоненційно.

Головною перевагою є фізична апаратна прив’язка — секрет зберігається розподілено, частина на USB-ключі зашифрована і не дає змоги відновити ключ без додаткових факторів. Це знижує ймовірність компрометації

$$P_{success} \to 0,$$

а ефективність захисту

$$E = 1 - P_{success} \to 100\%,$$

Отже, запропонований метод забезпечує суттєве підвищення безпеки за рахунок мультиплікативного збільшення складності атаки і адаптивного підходу до кількості факторів, що значно покращує загальну стійкість системи.

3.5 Висновки з розділу

У третьому розділі представлено програмну реалізацію системи захисту програмного забезпечення з апаратною прив'язкою до USB-носіїв. Обґрунтовано вибір мови програмування Python та бібліотеки tkinter для побудови графічного інтерфейсу. Детально описано архітектуру системи, що складається з восьми модулів: криптографічного модуля, схеми Шаміра, менеджерів USB та HWID, TOTP-генератора, калькулятора рівня довіри, менеджера бази даних та контролера захисту.

Розглянуто реалізацію криптографічних алгоритмів: генерацію 256-бітних ключів через secrets.token_bytes(), хешування паролів за PBKDF2-HMAC-SHA256, операції схеми Шаміра з використанням простого числа Мерсенна та

алгоритм TOTP відповідно до RFC 6238. Описано механізм адаптивної автентифікації з динамічним визначенням порогу k на основі показника довіри.

Результати тестування підтверджують коректність роботи всіх компонентів системи. Розроблений тестовий набір з 18 автоматизованих тестів охоплює позитивні та негативні сценарії, перевіряючи криптографічні операції, схему Шаміра, механізми автентифікації та стійкість до типових атак. Усі тести успішно пройдено із загальним часом виконання 0.221 секунди.

Представлено графічний інтерфейс користувача з функціональними панелями автентифікації, USB-пристроїв, захисту файлів та показника довіри. Описано структуру бази даних SQLite з п'яти таблиць для зберігання користувачів, USB-пристроїв, часток Шаміра, журналу автентифікації та захищених файлів.

4 ЕКОНОМІЧНА ЧАСТИНА

Магістерська кваліфікаційна робота на тему «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями [27].

За результатами розрахунків, наведених в таблиці 4.1, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації [27]. Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв» становить 42,3 бала, що свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

Таблиця 4.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Бали		
	Лебідь С.В.	Чухраєва О.В.	Стороженко А.С.
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	4	4	3
3. Ринкові переваги (ціна продукту)	4	4	3
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	4	4	4
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	2	2	2
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	2	3	2
10. Практична здійсненність (необхідність нових матеріалів)	3	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	42	44	41
Середньоарифметична сума балів $СБ_с$	42,3		

4.2 Розрахунок узагальненого коефіцієнта якості розробки

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення розрахуємо за формулою [28]:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i \,, \tag{4.1}$$

де k – кількість найбільш важливих технічних показників, які впливають на якість нового технічного рішення;

α_i – коефіцієнт, який враховує питому вагу *і-го* технічного показника в загальній якості розробки. Коефіцієнт α_i визначається експертним шляхом і при цьому має виконуватись умова $\sum_{i=1}^{k} \alpha_i = 1$;

β_i – відносне значення *і-го* технічного показника якості нової розробки.

Таблиця 4.2 – Порівняння основних параметрів розробки та аналога.

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований програмний засіб	Відношення параметрів нової розробки до аналога	Питома вага показника
Кількість факторів авторизації	од.	2	5	2,5	0,15
Дружність інтерфейсу інсталяції	бал	6	9	1,5	0,2
Швидкодія алгоритма	бал	7	9	1,28	0,3
Ймовірність відновлення втрачених даних	%	91	95	1,04	0,2
Рівень використання адаптивної оцінки довіри	%	50	98	1,96	0,15

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення складе:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i = 2{,}5 \cdot 0{,}15 + 1{,}5 \cdot 0{,}2 + 1{,}28 \cdot 0{,}3 + 1{,}04 \cdot 0{,}2 + 1{,}96 \cdot 0{,}15 = 1{,}56.$$

Отже за технічними параметрами наведеними в талиці 4.2, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,56 рази.

4.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

4.3.1 Витрати на оплату праці

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [27]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (4.4)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=22 дні.

$$З_о = 24600{,}00 \cdot 22 / 22 = 24600{,}00 \; грн.$$

Проведені розрахунки зведемо до таблиці 4.3.

Таблиця 4.3 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник науково-дослідної роботи (проєктний менеджер)	24600,00	1118,18	22	24600,00
Аналітик з розробки систем безпеки 1-ї кат.	22800,00	1036,36	7	7254,55
Інженер-програміст	21750,00	988,64	22	21750,00
Технік 1-ї категорії	9500,00	431,82	11	4750,00
Всього				58354,55

Основна заробітна плата робітників. Витрати на основну заробітну плату робітників ($З_р$) розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \,, \qquad (4.5)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \,, \qquad (4.6)$$

де M_M – розмір мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=8000,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення [27];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань.

T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дн;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 8000{,}00 \cdot 1{,}10 \cdot 1{,}15 / (22 \cdot 8) = 57{,}50$ *грн.*
$З_{р1} = 57{,}50 \cdot 6{,}00 = 345{,}00$ *грн.*

Проведені розрахунки зведемо до таблиці 4.4.

Таблиця 4.4 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка обчислювального обладнання для проведення розробки та досліджень	6,00	2	1,10	57,50	345,00
Підготовка робочого місця розробника програмного забезпечення	4,50	3	1,35	70,57	317,56
Інсталяція програмного забезпечення розробки засобу захисту програмного забезпечення	5,00	4	1,50	78,41	392,05
Введення дослідних баз даних ресурсу	12,00	3	1,35	70,57	846,82
Ведення кодів модульних блоків програмного забезпечення системи захисту	18,00	3	1,35	70,57	1270,23
Компіляція програмних модулів системи захисту програмного забезпечення	7,00	5	1,70	88,86	622,05
Налагодження блоків програмно-апаратного захисту програмного забезпечення	10,00	6	2,00	104,55	1045,45
Тестування засобу захисту програмного забезпечення з прив'язкою до USB-носіїв	5,00	2	1,10	57,50	287,50
Контроль протікання цифрового експерименту	6,50	4	1,50	78,41	509,66
Всього					5636,31

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.7)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (58354,55 + 5636,31) · 11 / 100% = 7038,99 грн.

4.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.8)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (58354,55 + 5636,31 + 7038,99) · 22 / 100% = 15626,57 грн.

4.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали тощо. Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (4.9)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

$В_j$ – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1$ = 2,000 · 185,00 · 1,05 - 0 · 0 = 388,50 грн.

Проведені розрахунки зведемо до таблиці 4.5.

Таблиця 4.5 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, од.	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Багатофункціональний білий офісний папір FAXE-500 A4	185,00	2,000	0	0	388,50
Папір для записів FAXE 70 A5-250	92,00	4,000	0	0	386,40
Органайзер офісний OFFICE 100	178,00	4,000	0	0	747,60
Набір офісний DATUM X-2	265,00	4,000	0	0	1113,00
Картридж для принтера HP-2100	1850,00	2,000	0	0	3885,00
Диск оптичний OPTIMA CD	32,00	4,000	0	0	134,40
USB Flesh-пам'ять GOODRAM 128 C10A	312,00	1,000	0	0	327,60
Тека для паперів CARBONIX BOX-ZX	190,00	4,000	0	0	798,00
Всього					7780,50

4.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв», розраховуємо, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \qquad (4.10)$$

де $Н_j$ – кількість комплектуючих *j*-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих *j*-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ *= 1,1 ... 1,15*).

$К_в = 1 \cdot 209,00 \cdot 1,05 = 219,45$ грн.

Проведені розрахунки зведемо до таблиці 4.6.

Таблиця 4.6 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Флеш USB Kingston DataTraveler Exodia 64GB USB 3.2 Black-Teal (DTX/64GB)	1	209,00	219,45
Флеш USB SanDisk Ultra Luxe 128GB USB (SDCZ74-128G-G46)	1	599,00	628,95
Флеш USB Apacer AH355 32GB USB 3.0 Black (AP32GAH355B-1)	1	199,00	208,95
Флеш USB MediaRange USB 3.0 combo flash drive, with USB Type-C 64GB Silver (MR937)	1	379,00	397,95
Всього			1455,30

4.3.5 Спецустаткування для наукових (експериментальних) робіт

Витрати за статтею «Спецустаткування» відсутні.

4.3.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення тощо. Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.11)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 439,00 \cdot 1 \cdot 1,05 = 460,95$ *грн.*

Отримані результати зведемо до таблиці 4.7:

Таблиця 4.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Доступ до мережі Internet (високошвидкісний) грн/місяць	1	439,00	460,95
Система імітаційного комп'ютерного моделювання SIMULINK програмного середовища MATLAB	1	5600,00	5880,00
Система керування реляційними базами даних MySQL (підписка)	1	320,00	336,00
Інтегроване середовище розробки Visual Studio 2022	1	6200,00	6510,00
Всього			13186,95

4.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.12)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (45899{,}00 \cdot 1) / (4 \cdot 12) = 956{,}23$ *грн.*

Проведені розрахунки зведемо до таблиці 4.8.

Таблиця 4.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер аналітичного дослідження ПК AMD Ryzen 9 5900X / NVIDIA RTX 5060 / 32 GB DDR4 / 1 TB SSD	45899,00	4	1	956,23
Принтер для ч/б друку HP LaserJet Tank 2502dw	14599,00	4	1	304,15
Робоче місце інженера-програміста	7700,00	5	1	128,33
Робоче місце аналітика систем безпеки	8990,00	5	1	149,83
Ноутбук Victus 16-r0018ua (BF1L8EA) Mica Silver	42999,00	3	1	1194,42
ОС Windows 11	8460,00	3	1	235,00
Microsoft Office 2019	7840,00	3	1	217,78
Маршрутизатор Asus ZenWiFi AX Mini XD4 3PK	9029,00	4	1	188,10
Приміщення лабораторії досліджень	650000,00	30	1	1805,56
Оргтехніка	12800,00	5	1	213,33
Всього				5392,73

4.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (4.13)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 12,56 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i<1$.

$B_e = 0,36 \cdot 172,0 \cdot 12,56 \cdot 0,95 / 0,97 = 777,72$ *грн.*

Проведені розрахунки зведемо до таблиці 4.9.

Таблиця 4.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер аналітичного дослідження ПК AMD Ryzen 9 5900X / NVIDIA RTX 5060 / 32 GB DDR4 / 1 TB SSD	0,36	172,0	777,72
Принтер для ч/б друку HP LaserJet Tank 2502dw з Wi-Fi (2R3E3A)	0,25	4,0	12,56
Робоче місце інженера-програміста	0,07	172,0	151,22
Робоче місце аналітика систем безпеки	0,07	172,0	151,22
Ноутбук Victus 16-r0018ua (BF1L8EA) Mica Silver	0,08	172,0	172,83
Маршрутизатор Asus ZenWiFi AX Mini XD4 3PK	0,02	172,0	43,21
Оргтехніка	0,56	2,0	14,07
Всього			1322,82

4.3.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв» належать витрати на відрядження штатних працівників тощо. Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%} \qquad (4.14)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 0%.

$B_{св} = (58354,55 + 5636,31) \cdot 0 / 100\% = 0,00$ *грн.*

4.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (4.15)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$B_{сп}$ = (58354,55 + 5636,31) · 30 / 100% = 19197,26 грн.

4.3.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат. Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.16)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (58354,55 + 5636,31) · 50 / 100% = 31995,43 грн.

4.3.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією тощо. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.17)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$B_{нзв}$ = (58354,55 + 5636,31) · 100 / 100% = 63990,85 грн.

Витрати на проведення науково-дослідної роботи на тему «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}. \quad (4.18)$$

$B_{заг}$= 58354,55 +5636,31 +7038,99 +15626,57 +7780,50 +1455,30 +0,00 + 13186,95 + 5392,73 +1322,82 +0,00 +19197,26 +31995,43 +63990,85 = 230978,24 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (4.19)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 230978,24 / 0,9 = 256642,49 грн.

4.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

Результати дослідження проведені за темою «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв» передбачають комерціалізацію протягом 4-х років реалізації на ринку, що наведено в таблиці 4.10.

Таблиця 4.10 – Витрати на електроенергію

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	4000	8000	5000	2000

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 200000 осіб;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 210,00 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 67,81 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, розраховуємо за формулою [27]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (4.20)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2025 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =40%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2025 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (67,81 \cdot 200000,00 + 277,81 \cdot 4000) \cdot 0,83 \cdot 0,4 \cdot (1-0,18/100\%) = 3994642,86$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (67,81 \cdot 200000,00 + 277,81 \cdot 12000) \cdot 0,83 \cdot 0,4 \cdot (1-0,18/100\%) = 4599690,81$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (67,81 \cdot 200000,00 + 277,81 \cdot 17000) \cdot 0,83 \cdot 0,4 \cdot (1-0,18/100\%) = 4977845,78$ грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4 = (67,81 \cdot 200000,00 + 277,81 \cdot 19000) \cdot 0,83 \cdot 0,4 \cdot (1-0,18/100\%) = 5129107,77$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (4.21)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=0,12;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 3994642,86/(1+0,12)^1+4599690,81/(1+0,12)^2+4977845,78/(1+0,12)^3+$$
$$+5129107,77/(1+0,12)^4=3566645,41+3666845,35+3543132,31+3259640,72=$$
$$=14036263,79 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (4.22)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=1,5;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 256642,49 грн.

$PV = k_{інв} \cdot ЗВ = 1,5 \cdot 256642,49 = 384963,74$ грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \quad (4.23)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 14036263,79 грн;

PV – теперішня вартість початкових інвестицій, 384963,74 грн.

$E_{абс} = ПП - PV$ = 14036263,79 - 384963,74 = 13651300,04 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.24)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 13651300,04 грн;

PV – теперішня вартість початкових інвестицій, 384963,74 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+13651300,04/384963,74)^{1/4} = 1,46.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (4.25)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,3.

$\tau_{мін}$ = 0,11+0,3 = 0,41 < 1,46. Тобто інвестувати в науково-дослідну роботу за темою «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \qquad (4.26)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,46 = 0,69 р.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

4.5 Висновки з розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв» становить 42,3 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

При оцінюванні за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,56 рази.

Також термін окупності становить 0,69 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв».

ВИСНОВКИ

У даній магістерській кваліфікаційній роботі було проведено комплексний аналіз сучасних методів захисту програмного забезпечення, зокрема рішень з апаратною прив’язкою до пристроїв зберігання даних. Метою дослідження стало створення вдосконаленого методу та програмного засобу захисту програмного забезпечення, який забезпечує підвищений рівень стійкості до несанкціонованого використання, копіювання та компрометації у порівнянні з традиційними підходами.

На основі проведеного аналізу було встановлено, що класичні методи апаратної прив’язки, такі як Hardware ID або USB-ключі зі статичним секретом, використовують лише один фактор автентифікації та характеризуються наявністю єдиної точки відмови. Це обмежує їхню ефективність і призводить до відносно невисокої стійкості у разі компрометації апаратного ідентифікатора або фізичного доступу до носія.

У межах роботи було розроблено новий метод захисту, що поєднує апаратну прив’язку до USB-носіїв із багатофакторною та адаптивною автентифікацією. Запропонований підхід базується на розподіленому зберіганні секрету між кількома незалежними факторами та використанні механізму динамічної оцінки рівня довіри. Це дозволяє усунути єдину точку відмови та суттєво зменшити ймовірність компрометації всієї системи при зламі окремого компонента.

Проведена оцінка ефективності розробленого засобу показала, що при використанні мінімальної кількості факторів захисту рівень безпеки не поступається найкращим традиційним USB-ключам. Водночас при збільшенні кількості факторів складність несанкціонованого доступу зростає на десятки і сотні порядків у порівнянні з класичними методами. При цьому наявність контрольованого механізму відновлення доступу, реалізованого на основі розподілу секрету, дозволяє легітимному користувачу відновити доступ без зниження загального рівня безпеки системи. Зокрема, при використанні трьох факторів рівень криптостійкості зростає приблизно у 10^{38} разів, а при чотирьох і

п’яти факторах – у 10^{77} та 10^{115} разів відповідно, що практично виключає можливість несанкціонованого перебору секрету та водночас не перешкоджає відновленню доступу за наявності достатньої кількості легітимних факторів.

Таким чином, мета магістерської роботи була досягнута. Розроблений метод і програмний засіб забезпечують істотне підвищення рівня захисту програмного забезпечення за рахунок багатофакторної апаратної прив’язки та адаптивного підходу до безпеки. Отримані результати підтверджують надійність, практичну значущість та перспективність запропонованого рішення для використання у системах, що потребують високого рівня захисту від несанкціонованого доступу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Singhal A., Venkataramalingam S. Malware Analysis and Reverse Engineering: Unraveling the Digital Threat Landscape. SSRN, 2023. – URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4649754 (accessed: 12.09.2025).

2. Advanced EXE Multi Protection Against Reverse Engineering with Free Tools // ExamCollection Blog. – URL: https://www.examcollection.com/blog/advanced-exe-multi-protection-against-reverse-engineering-with-free-tools/ (accessed: 12.09.2025).

3. AFT India. Hardware Against Software Piracy (HASP) Dongle Key. – URL: https://www.aftindia.in/hasp-dongle.html (accessed: 13.09.2025).

4. Guardant. Guardant User's Manual. – URL: https://www.technotrade.ua/Uploads/Guardant/Documentation/Guide.pdf (accessed: 13.04.2025).

5. Eutron Infosecurity. Eutron SmartKey SDK. – URL: https://eutron-smartkey-sdk.updatestar.com/en (accessed: 14.09.2025).

6. Максимець Д. В. Розробка комбінованого методу захисту програмного забезпечення від несанкціонованого використання. Магістерська дисертація, Київський політехнічний інститут, 2018. – URL: https://ela.kpi.ua/bitstream/123456789/23123/1/Maksymec_magistr.pdf (дата звернення: 14.09.2025).

7. Thales Group. SafeNet eToken 5110 Series. – URL: https://cpl.thalesgroup.com/access-management/authenticators/pki-usb-authentication/etoken-5110-usb-token (accessed: 14.09.2025).

8. Thales Group. Sentinel License Development Kit (LDK). – URL: https://cpl.thalesgroup.com/software-monetization/license-development-kit-ldk (accessed: 14.09.2025).

9. Національний інститут стандартів і технологій США. Advanced Encryption Standard (AES). – URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf (accessed: 14.09.2025).

10. Lee, R.P. (2015). Binding Hardware and Software to Prevent Firmware Piracy. ACM Digital Library. – URL: https://dl.acm.org/doi/10.1145/2899015.2899029 (accessed: 17.09.2025).

11. Sciensoft. Activation Key. – URL: https://www.sciensoft.com/docs/eleckey/Activation%20Key.htm (дата accessed: 17.09.2025).

12. Device Control – Endpoint Protector. URL: https://www.endpointprotector.com/solutions/device-control (accessed: 17.09.2025).

13. USB protection – Sofpro. URL: https://www.sofpro.com/docs/pc-guard/8-usb-protection (accessed: 17.09.2025).

14. Storage Control – ThreatLocker. URL: https://www.threatlocker.com/platform/storage-control (accessed: 17.09.2025).

15. Advanced Encryption Standard - Wikipedia. URL: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard (accessed: 05.10.2025).

16. What is the Advanced Encryption Standard (AES)? TechTarget. URL: https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard (accessed: 05.10.2025).

17. Transition to Advanced Encryption Standard (AES). CISA, 2024. URL: https://www.cisa.gov/sites/default/files/2024-05/23_0918_fpic_AES-Transition-WhitePaper_Final_508C_24_0513.pdf (accessed: 05.10.2025).

18.What Is AES Encryption & Is It Really the Best in 2025? Safety Detectives, December 2024. URL: https://www.safetydetectives.com/blog/what-is-aes-encryption/ (accessed: 05.10.2025).

19.AES: Your 2024 Guide to the Advanced Encryption Standard. URL: https://www.trainingtraining.training/blog/aes-guide-encryption-standard (accessed: 05.10.2025).

20.SHA-2 - Wikipedia. URL: https://en.wikipedia.org/wiki/SHA-2 (accessed: 05.10.2025).

21.SHA-256 Algorithm: Characteristics, Steps, and Applications. Simplilearn, June 2025. URL: https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm (accessed: 05.10.2025).

22.SHA-256 vs. SHA-1: Which Is The Better Encryption? SecureW2, September 2024. URL: https://www.securew2.com/blog/what-is-sha-encryption-sha-256-vs-sha-1 (accessed: 05.10.2025).

23.Is SHA-256 secure? Legal & Compliance Experts Say Yes. Page Freezer. URL: https://blog.pagefreezer.com/sha-256-benefits-evidence-authentication (accessed: 05.10.2025).

24.Rameel, Muhammad; Asif, Zain. Fortifying Information Security: a Comparative Analysis of AES, DES, 3DES, RSA, and Blowfish Algorithm. EasyChair Preprint № 13536. URL: https://easychair.org/publications/preprint/Kjbz/open (accessed: 06.10.2025).

25.Shamir A. How to share a secret. – Communications of the ACM. – 1979. – Vol. 22, No. 11. – P. 612–613. – DOI: 10.1145/359168.359176. – URL: https://doi.org/10.1145/359168.359176 (accessed: 21.10.2025).

26. Лесько О. В., Лукічов В. В.. Метод захисту програмного забезпечення з апаратною прив’язкою до USB-носіїв. Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2026)», Вінниця, 22-26 травня 2026 р. Електрон. текст. дані. 2026. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/view/26506 (дата звернення: 23.10.2025).

27. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с (дата звернення: 24.10.2025).

28. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с. (дата звернення: 24.10.2025).

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Метод та засіб захисту програмного застосунку з апаратною прив'язкою до USB-носіїв

Автор роботи: Лесько Олексій Віталійович
Тип роботи: магістерська кваліфікаційна робота
Підрозділ кафедра захисту інформації ФІТКІ , група 1 БС-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism **0,75** %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

√ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

В. о. зав. кафедри ЗІ д. т. н., проф. _____ Володимир ЛУЖЕЦЬКИЙ

Гарант освітньої програми «Безпека інформаційних та комунікаційних систем» к.т.н., доцент _____ Олеся ВОЙТОВИЧ

Особа, відповідальна за перевірку _____ Валентина КАПЛУН

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Віталій ЛУКІЧОВ

Здобувач _____ Олексій ЛЕСЬКО

ДОДАТОК Б

Фрагмент лістингу програмного застосунку

main.py

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog, scrolledtext
import sqlite3
import hashlib
import hmac
import struct
import base64
import time
import datetime
import os
import sys
import secrets
import threading
import json
import platform
import subprocess
import random
from pathlib import Path
from typing import List, Dict, Tuple, Optional
from dataclasses import dataclass
from enum import Enum

DB_PATH = "protection_module.db"
SHAMIR_PRIME = 2**521 - 1
TOTP_INTERVAL = 30
TOTP_DIGITS = 6
PASSWORD_ITERATIONS = 100000
SALT_SIZE = 16
KEY_SIZE = 32

class AuthStatus(Enum):
    SUCCESS = "success"
    FAILED = "failed"
    LOCKED = "locked"
    RECOVERY_REQUIRED = "recovery_required"

@dataclass
class UsbDevice:
    serial: str
    vendor_id: str
    product_id: str
    name: str
    mount_point: str = ""

@dataclass
class ShamirShare:
    x: int
    y: int
    factor_type: str

@dataclass
class TrustScore:
    usb_factor: float
    time_factor: float
    location_factor: float
    attempts_factor: float
    total: float
    required_k: int



class UsbDeviceManager:
```

```
 def __init__(self):
        self.registered_devices: Dict[str, UsbDevice] = {}
        self.current_devices: List[UsbDevice] = []

    def scan_usb_devices(self) -> List[UsbDevice]:
        devices = []
        system = platform.system()

        if system == "Windows":
            devices = self._scan_windows()
        elif system == "Linux":
            devices = self._scan_linux()
        elif system == "Darwin":
            devices = self._scan_macos()

        self.current_devices = devices
        return devices

    def _scan_linux(self) -> List[UsbDevice]:
        devices = []
        usb_path = Path("/sys/bus/usb/devices")

        if not usb_path.exists():
            return self._generate_demo_devices()

        try:
            for device_dir in usb_path.iterdir():
                if not device_dir.is_dir():
                    continue

                serial_path = device_dir
                vendor_path = device_dir
                product_path = device_dir
                manufacturer_path = device_dir
                product_name_path = device_dir

                if serial_path.exists() and vendor_path.exists():
                    try:
                        serial = serial_path.read_text().strip()
                        vendor_id = vendor_path.read_text().strip()
                        product_id = product_path.read_text().strip() if
product_path.exists() else "0000"

                        name = "USB Device"
                        if product_name_path.exists():
                            name = product_name_path.read_text().strip()
                        elif manufacturer_path.exists():
                            name = manufacturer_path.read_text().strip()

                        if serial and len(serial) > 4:
                            devices.append(UsbDevice(
                                serial=serial,
                                vendor_id=vendor_id,
                                product_id=product_id,
                                name=name
                            ))
                    except (PermissionError, IOError):
                        continue
        except Exception:
            pass

        if not devices:
            devices = self._generate_demo_devices()

        return devices

    def _scan_windows(self) -> List[UsbDevice]:
        """Сканування USB на Windows через WMI"""
        devices = []
        try:
```

```
        try:


            import wmi
        except ImportError:
            raise ImportError("WMI module not available")
        c = wmi.WMI()
        for usb in c.Win32_USBHub():
            if usb.DeviceID:
                parts = usb.DeviceID.split("\\")
                if len(parts) >= 3:
                    vid_pid = parts[1] if len(parts) > 1 else ""
                    serial = parts[2] if len(parts) > 2 else secrets.token_hex(8)

                    vendor_id = "0000"
                    product_id = "0000"
                    if "VID_" in vid_pid and "PID_" in vid_pid:
                        try:
                            vendor_id = vid_pid.split("VID_")[1][:4]
                            product_id = vid_pid.split("PID_")[1][:4]
                        except:
                            pass

                    devices.append(UsbDevice(
                        serial=serial,
                        vendor_id=vendor_id,
                        product_id=product_id,
                        name=usb.Description or "USB Device"
                    ))
    except ImportError:
        devices = self._generate_demo_devices()
    except Exception:
        devices = self._generate_demo_devices()

    if not devices:
        devices = self._generate_demo_devices()

    return devices

def _scan_macos(self) -> List[UsbDevice]:
    devices = []
    try:
        result = subprocess.run(
            ["system_profiler", "SPUSBDataType", "-json"],
            capture_output=True, text=True, timeout=10
        )
        if result.returncode == 0:
            data = json.loads(result.stdout)
            self._parse_macos_usb(data, devices)
    except Exception:
        pass

    if not devices:
        devices = self._generate_demo_devices()

    return devices

def _parse_macos_usb(self, data: dict, devices: list, depth=0):
    if depth > 10:
        return
    if isinstance(data, dict):
        if "serial_num" in data:
            devices.append(UsbDevice(
                serial=data.get("serial_num", secrets.token_hex(8)),
                vendor_id=data.get("vendor_id", "0000"),
                product_id=data.get("product_id", "0000"),
                name=data.get("_name", "USB Device")
            ))
        for value in data.values():
            self._parse_macos_usb(value, devices, depth+1)
```

```
        elif isinstance(data, list):
            for item in data:
                self._parse_macos_usb(item, devices, depth+1)

        def get_device_hash(self, device: UsbDevice) -> str:
        data = f"{device.serial}:{device.vendor_id}:{device.product_id}"
        return hashlib.sha256(data.encode()).hexdigest()

    def is_device_registered(self, device: UsbDevice) -> bool:
        device_hash = self.get_device_hash(device)
        return device_hash in self.registered_devices

    def register_device(self, device: UsbDevice):
        device_hash = self.get_device_hash(device)
        self.registered_devices[device_hash] = device

class HwidManager:

    def __init__(self):
        self.current_hwid = self.collect_hwid()

    def collect_hwid(self) -> str:
        components = []

        try:
            if platform.system() == "Linux":
                with open("/proc/cpuinfo", "r") as f:
                    for line in f:
                        if "model name" in line or "Serial" in line:
                            components.append(line.strip())
                            break
            elif platform.system() == "Windows":
                import subprocess
                result = subprocess.run(
                    ["wmic", "cpu", "get", "processorid"],
                    capture_output=True, text=True
                )
                components.append(result.stdout.strip())
        except:
            components.append(f"cpu_{platform.processor()}")
        try:
            if platform.system() == "Linux":
                machine_id_path = Path("/etc/machine-id")
                if machine_id_path.exists():
                    components.append(machine_id_path.read_text().strip())
        except:
            pass


        components.append(platform.node())
        components.append(platform.system())
        components.append(platform.machine())


        combined = "|".join(components)
        return hashlib.sha256(combined.encode()).hexdigest()

    def verify_hwid(self, stored_hwid: str) -> bool:
        return self.current_hwid == stored_hwid

    def get_hwid_short(self) -> str:
        return f"{self.current_hwid[:8]}...{self.current_hwid[-8:]}"


class TotpManager:

    def __init__(self, secret: str = TOTP_SECRET, interval: int = TOTP_INTERVAL, digits:
int = TOTP_DIGITS):
        self.secret = secret
        self.interval = interval
```

```
        self.digits = digits

    def _decode_secret(self) -> bytes:
        s = self.secret.upper()
        padding = 8 - (len(s) % 8)
        if padding != 8:
            s += "=" * padding
        return base64.b32decode(s)

    def _get_counter(self, timestamp: float = None) -> int:
        if timestamp is None:
            timestamp = time.time()
        return int(timestamp // self.interval)

    def generate(self, counter: int = None) -> str:
        if counter is None:
            counter = self._get_counter()

        key = self._decode_secret()
        msg = struct.pack(">Q", counter)
        h = hmac.new(key, msg, hashlib.sha1).digest()

        offset = h[-1] & 0x0F
        code = (
            ((h[offset] & 0x7F) << 24) |
            ((h[offset + 1] & 0xFF) << 16) |
            ((h[offset + 2] & 0xFF) << 8) |
            (h[offset + 3] & 0xFF)
        )

        otp = code % (10 ** self.digits)
        return f"{otp:0{self.digits}d}"

    def current(self) -> str:
        return self.generate()

    def verify(self, code: str, window: int = 1) -> bool:
        try:
            int(code)
        except ValueError:
            return False

        counter = self._get_counter()
        for delta in range(-window, window + 1):
            if self.generate(counter + delta) == code:
                return True
        return False



 def time_remaining(self) -> int:
        return self.interval - (int(time.time()) % self.interval)
class ShamirSecretSharing:

    def __init__(self, prime: int = SHAMIR_PRIME):
        self.prime = prime

    def _mod_inverse(self, a: int, p: int) -> int:
        def extended_gcd(a, b):
            if a == 0:
                return b, 0, 1
            gcd, x1, y1 = extended_gcd(b % a, a)
            x = y1 - (b // a) * x1
            y = x1
            return gcd, x, y

        _, x, _ = extended_gcd(a % p, p)
        return (x % p + p) % p

    def generate_shares(self, secret: int, n: int, k: int) -> List[ShamirShare]:
```

```
        if k > n:
            raise ValueError("Поріг k не може перевищувати кількість часток n")
        if secret >= self.prime:
            raise ValueError("Секрет повинен бути меншим за просте число")


        coefficients = [secret]
        for _ in range(k - 1):
            coefficients.append(secrets.randbelow(self.prime))


        shares = []
        factor_types = ["USB", "HWID", "Password", "TOTP", "Recovery"]

        for x in range(1, n + 1):
            y = 0
            for i, coef in enumerate(coefficients):
                y = (y + coef * pow(x, i, self.prime)) % self.prime
            shares.append(ShamirShare(x=x, y=y, factor_type=factor_types[x-1] if x <= 5
else f"Factor_{x}"))

        return shares

    def reconstruct_secret(self, shares: List[ShamirShare]) -> int:
        if len(shares) < 2:
            raise ValueError("Потрібно мінімум 2 частки для відновлення")

        secret = 0
        for i, share_i in enumerate(shares):
            numerator = 1
            denominator = 1

            for j, share_j in enumerate(shares):
                if i != j:
                    numerator = (numerator * (-share_j.x)) % self.prime
                    denominator = (denominator * (share_i.x - share_j.x)) % self.prime

            lagrange = (numerator * self._mod_inverse(denominator, self.prime)) %
self.prime
            secret = (secret + share_i.y * lagrange) % self.prime

        return secret

    def verify_reconstruction(self, original: int, shares: List[ShamirShare], k: int)
bool:
        if len(shares) < k:
            return False


 try:

  reconstructed = self.reconstruct_secret(shares[:k])
            return reconstructed == original
        except:
            return False

class CryptoModule:

    @staticmethod
    def generate_master_key() -> bytes:
        return secrets.token_bytes(KEY_SIZE)

    @staticmethod
    def generate_salt() -> bytes:
        return secrets.token_bytes(SALT_SIZE)


    @staticmethod
    def derive_key_from_password(password: str, salt: bytes) -> bytes:
        return hashlib.pbkdf2_hmac(
```

```
            'sha256',
            password.encode('utf-8'),
            salt,
            PASSWORD_ITERATIONS,
            dklen=KEY_SIZE
        )

    @staticmethod
    def hash_password(password: str, salt: bytes = None) -> Tuple[bytes, bytes]:
        if salt is None:
            salt = CryptoModule.generate_salt()
        key = CryptoModule.derive_key_from_password(password, salt)
        return key, salt

    @staticmethod
    def verify_password(password: str, salt: bytes, stored_hash: bytes) -> bool:

        derived = CryptoModule.derive_key_from_password(password, salt)
        return hmac.compare_digest(derived, stored_hash)


 @staticmethod
    def hash_recovery_phrase(phrase: str) -> bytes:
        return hashlib.pbkdf2_hmac(
            'sha512',
            phrase.encode('utf-8'),
            b'mnemonic',
            2048,
            dklen=64
        )
class TrustScoreCalculator:


    def __init__(self):
        self.weights = {
            'usb': 0.4,
            'time': 0.3,
            'location': 0.2,
            'attempts': 0.1


    def calculate_usb_factor(self, usb_present: bool, usb_history: int) -> float:
        if not usb_present:
            return 0.3
        if usb_history > 10:
            return 1.0
        elif usb_history > 5:
            return 0.8
        elif usb_history > 0:
            return 0.5
        return 0.5

    def calculate_time_factor(self, hour: int, day_of_week: int, history: Dict = None) ->
float:
        if 9 <= hour <= 18 and 0 <= day_of_week <= 4:
            return 1.0
        elif 7 <= hour <= 22:
            return 0.7
        else:
            return 0.4

    def calculate_location_factor(self, ip_known: bool, same_country: bool) -> float:

        if ip_known:
            return 1.0
        elif same_country:
            return 0.6
        return 0.2

    def calculate_attempts_factor(self, failed_attempts: int) -> float:
```

```
    if failed_attempts == 0:
        return 1.0
    elif failed_attempts == 1:
        return 0.8
    elif failed_attempts == 2:
        return 0.5
    elif failed_attempts == 3:
        return 0.2
    return 0.0

def calculate(self, usb_present: bool, usb_history: int, failed_attempts: int,
              ip_known: bool = True, same_country: bool = True) -> TrustScore:
    now = datetime.datetime.now()

    usb_factor = self.calculate_usb_factor(usb_present, usb_history)
    time_factor = self.calculate_time_factor(now.hour, now.weekday())
    location_factor = self.calculate_location_factor(ip_known, same_country)
    attempts_factor = self.calculate_attempts_factor(failed_attempts)

    total = (
        self.weights['usb'] * usb_factor +
        self.weights['time'] * time_factor +
        self.weights['location'] * location_factor +
        self.weights['attempts'] * attempts_factor
    ) * 100

    if total >= 80:
        required_k = 2
    elif total >= 50:
        required_k = 3
    elif total >= 30:
        required_k = 4
    else:
        required_k = 5

    return TrustScore(
        usb_factor=usb_factor,
        time_factor=time_factor,
        location_factor=location_factor,
        attempts_factor=attempts_factor,
        total=total,
        required_k=required_k
    )

def __init__(self, db_path: str = DB_PATH):
    self.db_path = db_path
    self.conn = None
    self.init_database()

def init_database(self):
    self.conn = sqlite3.connect(self.db_path, check_same_thread=False)
    cursor = self.conn.cursor()

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            email TEXT UNIQUE NOT NULL,
            password_hash BLOB NOT NULL,
            password_salt BLOB NOT NULL,
            master_key_hash TEXT NOT NULL,
            hwid TEXT,
            totp_secret TEXT,
            recovery_phrase_hash BLOB,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            last_login TIMESTAMP,
            is_active INTEGER DEFAULT 1
        )
    ''')
```

```
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS usb_devices (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id INTEGER NOT NULL,
                device_hash TEXT NOT NULL,
                serial TEXT NOT NULL,
                vendor_id TEXT,
                product_id TEXT,
                device_name TEXT,
                registered_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                last_used TIMESTAMP,
                use_count INTEGER DEFAULT 0,
                is_active INTEGER DEFAULT 1,



        FOREIGN KEY (user_id) REFERENCES users(id)
            )
        ''')

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS shamir_shares (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id INTEGER NOT NULL,
                factor_type TEXT NOT NULL,
                x_value INTEGER NOT NULL,
                y_value_encrypted TEXT NOT NULL,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY (user_id) REFERENCES users(id)
            )
        ''')

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS auth_attempts (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id INTEGER,
                ip_address TEXT,
                success INTEGER NOT NULL,
                factors_used TEXT,
                trust_score REAL,
                required_k INTEGER,
                timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY (user_id) REFERENCES users(id)
            )
        ''')

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS protected_files (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id INTEGER NOT NULL,
                file_path TEXT NOT NULL,
                file_hash TEXT NOT NULL,
                protection_type TEXT DEFAULT 'full',
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                last_accessed TIMESTAMP,
                FOREIGN KEY (user_id) REFERENCES users(id)
            )
        ''')

        self.conn.commit()

    def add_user(self, email: str, password_hash: bytes, password_salt: bytes,
                 master_key_hash: str, hwid: str, totp_secret: str) -> int:
        cursor = self.conn.cursor()
        cursor.execute('''
            INSERT INTO users (email, password_hash, password_salt, master_key_hash,
hwid, totp_secret)
            VALUES (?, ?, ?, ?, ?, ?)
        ''', (email, password_hash, password_salt, master_key_hash, hwid, totp_secret))
        self.conn.commit()
```

```
        return cursor.lastrowid

    def get_user_by_email(self, email: str) -> Optional[dict]:
        cursor = self.conn.cursor()
        cursor.execute('SELECT * FROM users WHERE email = ?', (email,))
        row = cursor.fetchone()
        if row:
            columns = [desc[0] for desc in cursor.description]
            return dict(zip(columns, row))
        return None

    def add_usb_device(self, user_id: int, device: UsbDevice, device_hash: str):

        cursor = self.conn.cursor()
        cursor.execute('''
            INSERT INTO usb_devices (user_id, device_hash, serial, vendor_id, product_id,
device_name)
            VALUES (?, ?, ?, ?, ?, ?)
        ''', (user_id, device_hash, device.serial, device.vendor_id, device.product_id,
device.name))
        self.conn.commit()

    def get_user_usb_devices(self, user_id: int) -> List[dict]:
        cursor = self.conn.cursor()
        cursor.execute('SELECT * FROM usb_devices WHERE user_id = ? AND is_active = 1',
(user_id,))
        rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]
        return [dict(zip(columns, row)) for row in rows]

    def update_usb_usage(self, device_hash: str):
        cursor = self.conn.cursor()
        cursor.execute('''
            UPDATE usb_devices
            SET last_used = CURRENT_TIMESTAMP, use_count = use_count + 1
            WHERE device_hash = ?
        ''', (device_hash,))
        self.conn.commit()
```

test_protection.py

```
import unittest
import os
import sys
import hashlib
import secrets
import time
from datetime import datetime

from core import (
    CryptoModule, ShamirSecretSharing, ShamirShare,
    TotpManager, HwidManager, UsbDeviceManager, UsbDevice,
    TrustScoreCalculator, DatabaseManager,
    SHAMIR_PRIME, KEY_SIZE
)

class TestCryptoModule(unittest.TestCase):

    def test_master_key_generation(self):
        print("\n" + "="*60)
        print("="*60)
```

```
        key = CryptoModule.generate_master_key()
        self.assertEqual(len(key), KEY_SIZE, f"Очікувано {KEY_SIZE} байт")
        print(f"Розмір ключа: {len(key)} байт ({len(key)*8} біт)")

        key2 = CryptoModule.generate_master_key()
        self.assertNotEqual(key, key2, "Ключі мають бути унікальними")
        print(f"Унікальність підтверджена")

        unique_bytes = len(set(key))
        self.assertGreater(unique_bytes, 20, "Недостатня ентропія")
        print(f" Унікальних байтів: {unique_bytes}")
        print(f"\nКлюч (hex): {key.hex()[:64]}...")
        print("ТЕСТ ПРОЙДЕНО ")

    def test_password_hashing(self):
        print("\n" + "="*60)
        print("ТЕСТ 2: Хешування паролів (PBKDF2-HMAC-SHA256)")
        print("="*60)

        password = "SecurePassword123!"
        hash1, salt1 = CryptoModule.hash_password(password)
        hash2, salt2 = CryptoModule.hash_password(password)

        self.assertNotEqual(salt1, salt2)
        self.assertNotEqual(hash1, hash2)
        print(f" Унікальність солей підтверджена")

        self.assertTrue(CryptoModule.verify_password(password, salt1, hash1))
        self.assertFalse(CryptoModule.verify_password("WrongPassword", salt1, hash1))
        print(f" Верифікація працює")
        print(f"\nСіль: {salt1.hex()}")
        print(f"Хеш: {hash1.hex()[:64]}...")
        print("ТЕСТ ПРОЙДЕНО ")

class TestShamirSecretSharing(unittest.TestCase):

    def setUp(self):
        self.shamir = ShamirSecretSharing()

    def test_share_generation(self):
        """Тест 3: Генерація 5 часток секрету"""
        print("\n" + "="*60)
        print("ТЕСТ 3: Генерація 5 часток за схемою Шаміра")


  print("="*60)

        secret = secrets.randbelow(SHAMIR_PRIME - 1) + 1
        shares = self.shamir.generate_shares(secret, n=5, k=2)

        self.assertEqual(len(shares), 5)
        print(f" Згенеровано {len(shares)} часток")

        factor_types = [s.factor_type for s in shares]
        expected = ["USB", "HWID", "Password", "TOTP", "Recovery"]
        self.assertEqual(factor_types, expected)
        print(f" Типи факторів: {factor_types}")

        for i, share in enumerate(shares):
            print(f"  Частка {i+1}: x={share.x}, тип={share.factor_type}")
        print("ТЕСТ ПРОЙДЕНО ")


class TestTOTP(unittest.TestCase):
    def setUp(self):
        self.totp = TotpManager()

    def test_totp_generation(self):
        print("\n" + "="*60)
```

```
        print("ТЕСТ 7: Генерація та верифікація TOTP")
        print("="*60)

        code = self.totp.current()
        self.assertEqual(len(code), 6)
        self.assertTrue(code.isdigit())
        print(f" Поточний TOTP: {code}")

        self.assertTrue(self.totp.verify(code))
        print(f" Верифікація: OK")

        self.assertFalse(self.totp.verify("000000"))
        print(f" Хибний код відхилено")
        print("ТЕСТ ПРОЙДЕНО")

    def test_totp_window(self):
        print("\n" + "="*60)
        print("ТЕСТ 8: Часове вікно TOTP")
        print("="*60)

        counter = self.totp._get_counter()
        previous = self.totp.generate(counter - 1)
        current = self.totp.generate(counter)
        next_code = self.totp.generate(counter + 1)

        print(f"  Попередній: {previous}")
        print(f"  Поточний:   {current}")
        print(f"  Наступний:  {next_code}")

        self.assertTrue(self.totp.verify(previous, window=1))
        self.assertTrue(self.totp.verify(current, window=1))
        self.assertTrue(self.totp.verify(next_code, window=1))
        print(f" Всі три проходять верифікацію")

        old = self.totp.generate(counter - 2)
        self.assertFalse(self.totp.verify(old, window=1))
        print(f" Застарілий відхилено")
        print("ТЕСТ ПРОЙДЕНО ")

class TestHWID(unittest.TestCase):
    def test_hwid_collection(self):
        print("\n" + "="*60)
        print("ТЕСТ 9: Збір та верифікація HWID")
        print("="*60)

        hwid = HwidManager()
        self.assertEqual(len(hwid.current_hwid), 64)
        print(f" HWID: {hwid.get_hwid_short()}")

        self.assertTrue(hwid.verify_hwid(hwid.current_hwid))
        print(f" Самоверифікація: OK")

        self.assertFalse(hwid.verify_hwid("a" * 64))
        print(f" Підробка відхилена")
        print("ТЕСТ ПРОЙДЕНО ")

class TestUSB(unittest.TestCase):
    def test_usb_scan(self):
        print("\n" + "="*60)
        print("ТЕСТ 10: Сканування USB пристроїв")
        print("="*60)



  usb = UsbDeviceManager()
        devices = usb.scan_usb_devices()

        self.assertIsInstance(devices, list)
        print(f" Знайдено пристроїв: {len(devices)}")
```

```
        for i, dev in enumerate(devices):
            print(f"  Пристрій {i+1}: {dev.serial[:20]}...")

        if devices:
            h = usb.get_device_hash(devices[0])
            self.assertEqual(len(h), 64)
            print(f"Хеш пристрою: {h[:32]}...")
        print("ТЕСТ ПРОЙДЕНО ")

class TestTrustScore(unittest.TestCase):
    def setUp(self):
        self.calc = TrustScoreCalculator()

    def test_normal_scenario(self):
        print("\n" + "="*60)
        print("ТЕСТ 11: Trust Score - Нормальний")
        print("="*60)

        score = self.calc.calculate(usb_present=True, usb_history=15, failed_attempts=0)

        print(f"  USB: {score.usb_factor:.2f}")
        print(f"  Time: {score.time_factor:.2f}")
        print(f"  Location: {score.location_factor:.2f}")
        print(f"  Attempts: {score.attempts_factor:.2f}")
        print(f"  TOTAL: {score.total:.1f}%")
        print(f"  k: {score.required_k}")

        self.assertGreaterEqual(score.total, 80)
        self.assertEqual(score.required_k, 2)
        print(f" k=2 при Trust ≥80%")
        print("ТЕСТ ПРОЙДЕНО")

    def test_anomaly_scenario(self):
        print("\n" + "="*60)
        print("ТЕСТ 12: Trust Score - Аномальний")
        print("="*60)

        score = self.calc.calculate(usb_present=False, usb_history=0, failed_attempts=1)

        print(f"  USB: {score.usb_factor:.2f} (відсутній)")
        print(f"  Attempts: {score.attempts_factor:.2f} (1 невдала)")
        print(f"  TOTAL: {score.total:.1f}%")
        print(f"  k: {score.required_k}")

        self.assertLess(score.total, 80)
        self.assertGreaterEqual(score.required_k, 3)
        print(f" k≥3 при Trust <80%")
        print("ТЕСТ ПРОЙДЕНО ")

    def test_risk_scenario(self):
        print("\n" + "="*60)
        print("ТЕСТ 13: Trust Score - Ризиковий")
        print("="*60)

        score = self.calc.calculate(usb_present=False, usb_history=0, failed_attempts=4,
                                    ip_known=False, same_country=False)



   print(f"  USB: {score.usb_factor:.2f}")
        print(f"  Location: {score.location_factor:.2f} (невідома)")
        print(f"  Attempts: {score.attempts_factor:.2f} (4+)")
        print(f"  TOTAL: {score.total:.1f}%")
        print(f"  k: {score.required_k}")

        self.assertLess(score.total, 50)
        self.assertGreaterEqual(score.required_k, 4)
        print(f" k≥4 при Trust <50%")
        print("ТЕСТ ПРОЙДЕНО")
```

```
class TestNegativeScenarios(unittest.TestCase):
    def test_missing_usb(self):
        print("\n" + "="*60)
        print("ТЕСТ 14: Негативний - Відсутність USB")
        print("="*60)

        calc = TrustScoreCalculator()
        with_usb = calc.calculate(usb_present=True, usb_history=10, failed_attempts=0)
        without_usb = calc.calculate(usb_present=False, usb_history=0, failed_attempts=0)

        print(f"  З USB: {with_usb.total:.1f}%, k={with_usb.required_k}")
        print(f"  Без USB: {without_usb.total:.1f}%, k={without_usb.required_k}")

        self.assertGreater(with_usb.total, without_usb.total)
        print(f" Відсутність USB збільшує k")
        print("ТЕСТ ПРОЙДЕНО ")

    def test_corrupted_shares(self):
        print("\n" + "="*60)
        print("ТЕСТ 15: Негативний - Пошкоджені частки")
        print("="*60)

        shamir = ShamirSecretSharing()
        secret = 123456789
        shares = shamir.generate_shares(secret, n=5, k=2)

        corrupted = [shares[0], ShamirShare(x=shares[1].x, y=shares[1].y + 1,
factor_type="HWID")]
        result = shamir.reconstruct_secret(corrupted)

        self.assertNotEqual(result, secret)
        print(f" Оригінал: {secret}")
        print(f" Пошкоджений: {result}")
        print(f" Секрет НЕ відновлено")



def run_all_tests():
    print("\n" + "="*70)
    print("="*70)
    print(f"Дата: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    print("="*70)

    loader = unittest.TestLoader()
    suite = unittest.TestSuite()

    suite.addTests(loader.loadTestsFromTestCase(TestCryptoModule))
    suite.addTests(loader.loadTestsFromTestCase(TestShamirSecretSharing))
    suite.addTests(loader.loadTestsFromTestCase(TestTOTP))
    suite.addTests(loader.loadTestsFromTestCase(TestHWID))
    suite.addTests(loader.loadTestsFromTestCase(TestUSB))
    suite.addTests(loader.loadTestsFromTestCase(TestTrustScore))
    suite.addTests(loader.loadTestsFromTestCase(TestNegativeScenarios))
    suite.addTests(loader.loadTestsFromTestCase(TestDatabase))

    runner = unittest.TextTestRunner(verbosity=0)
    result = runner.run(suite)
```

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОД ТА ЗАСІБ ЗАХИСТУ ПРОГРАМНОГО ЗАСТОСУНКУ З АПАРАТНОЮ ПРИВ'ЯЗКОЮ ДО USB-НОСІЇВ

Виконав: студент 2 курсу групи 1БС-24 м спеціальності 125 Кібербезпека та захист інформації

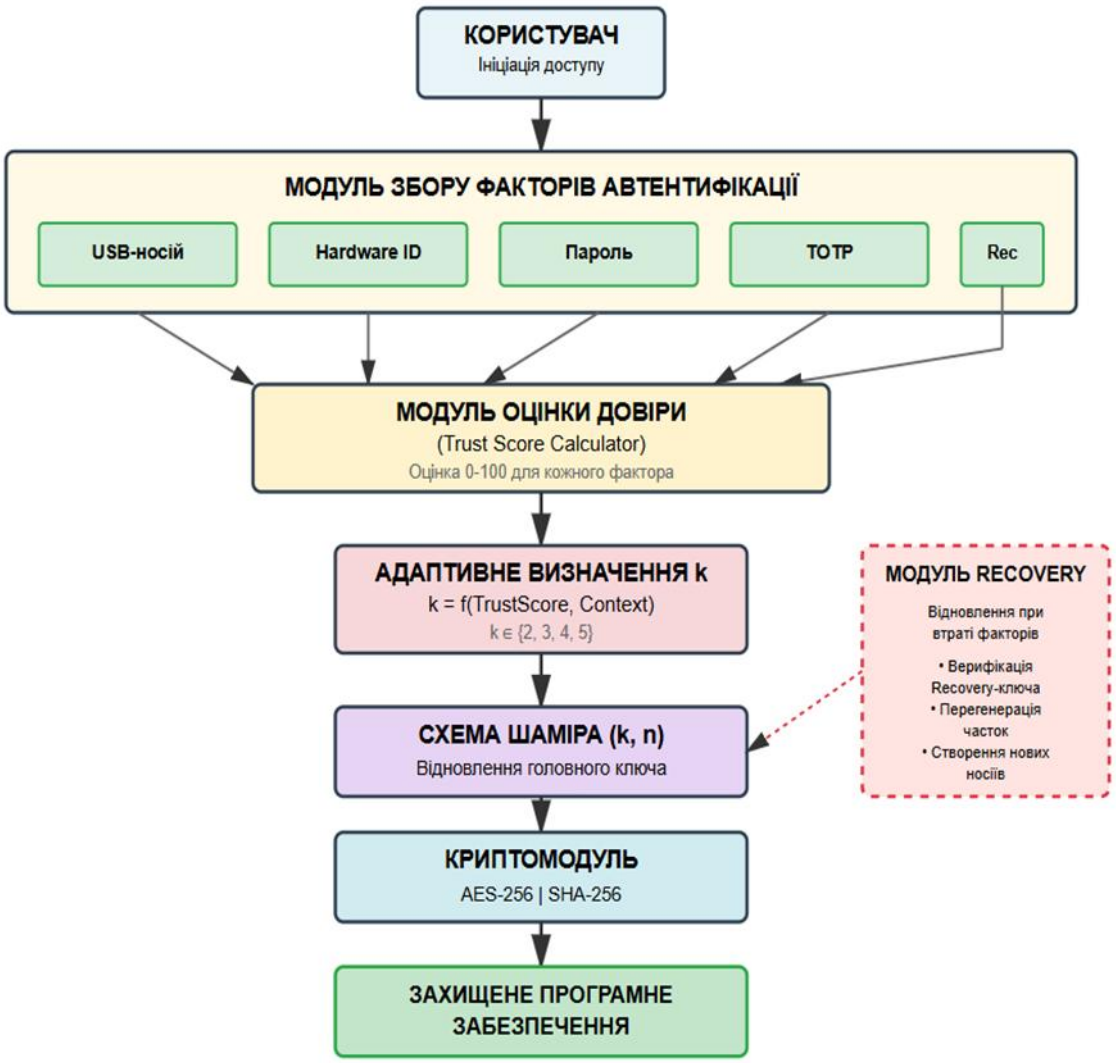
_____Олексій ЛЕСЬКО

Керівник: к. т. н., доц., доцент каф. ЗІ

_____Віталій ЛУКІЧОВ

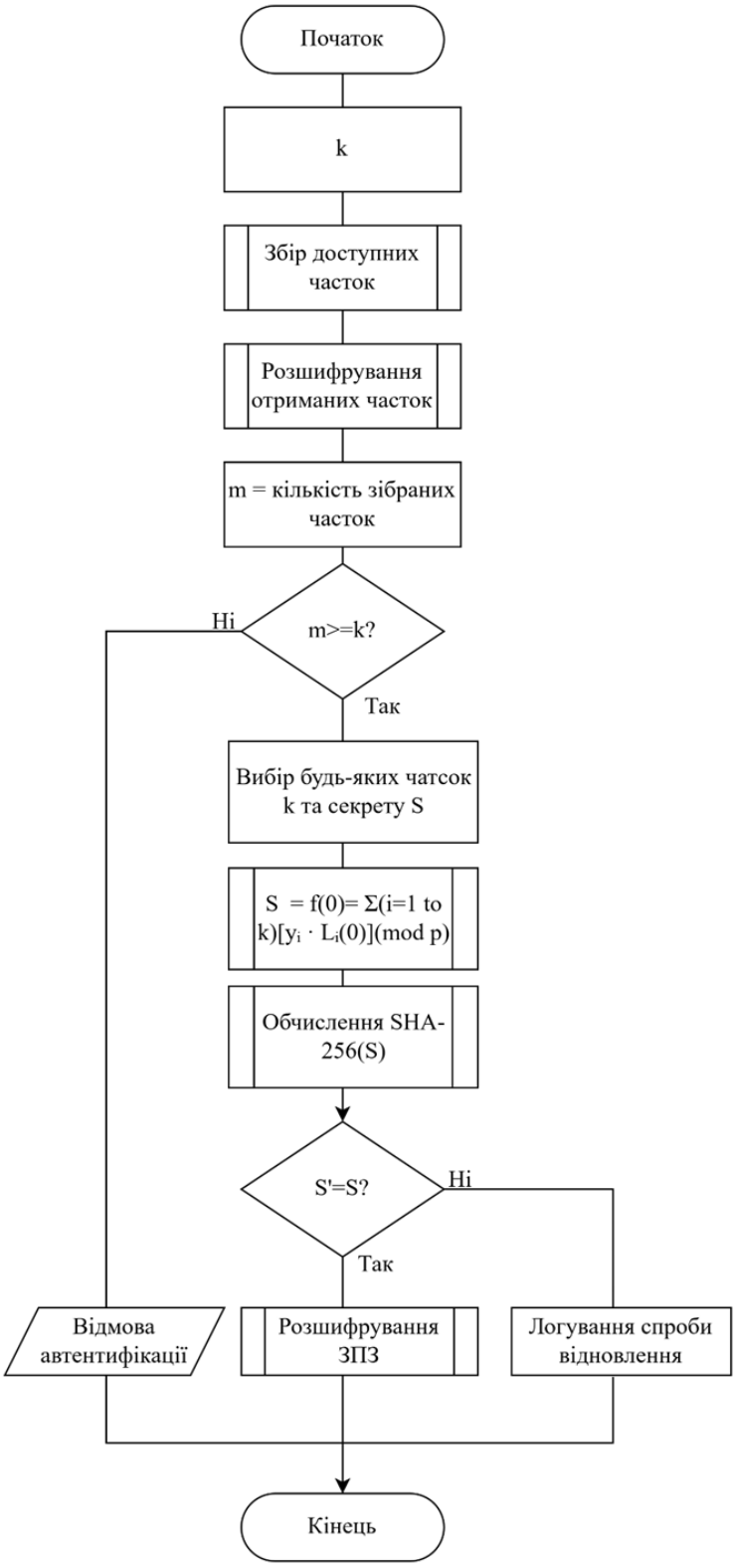
«19» грудня 2025 р.

АРХІТЕКТУРА СИСТЕМИ ЗАХИСТУ ПРОГРАМНОГО ЗАСТОСУНКУ

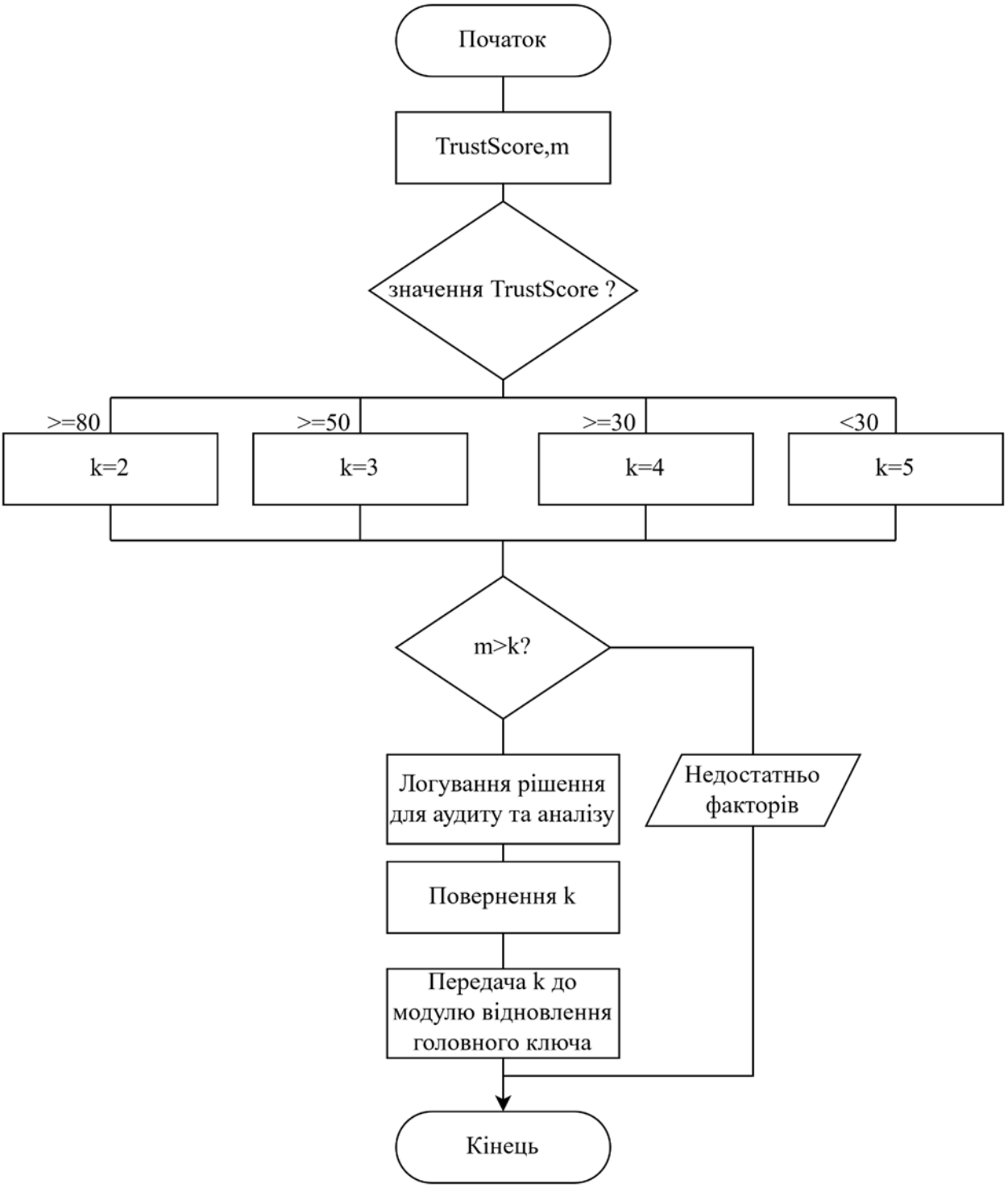


АЛГОРИТМ ГЕНЕРАЦІЇ ТА РОЗПОДІЛУ ЧАСТОК СЕКРЕТУ

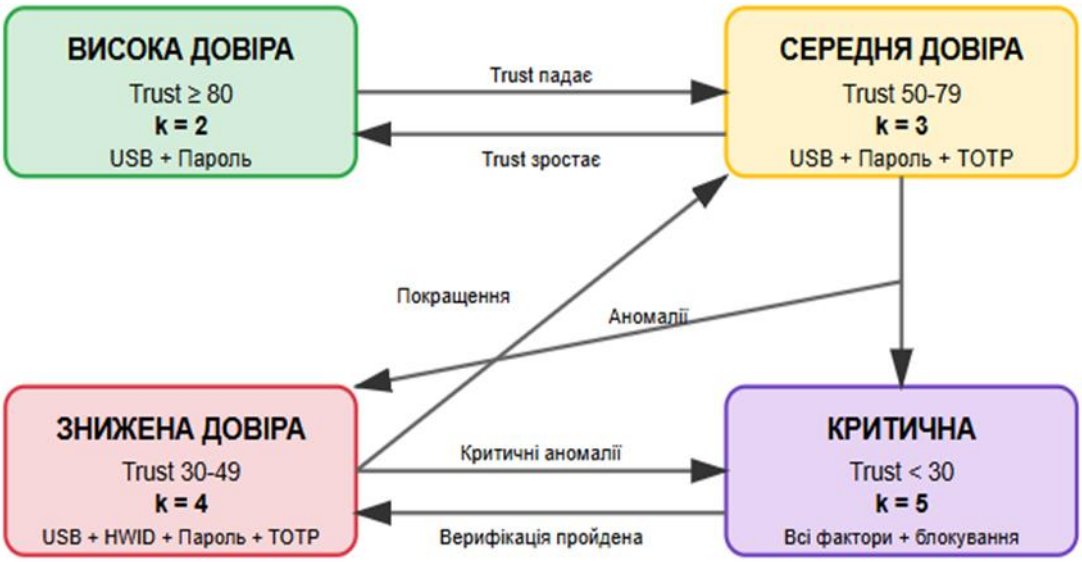
АЛГОРИТМ ВІДНОВЛЕННЯ СЕКРЕТУ ЗА СХЕМОЮ ШАМІРА



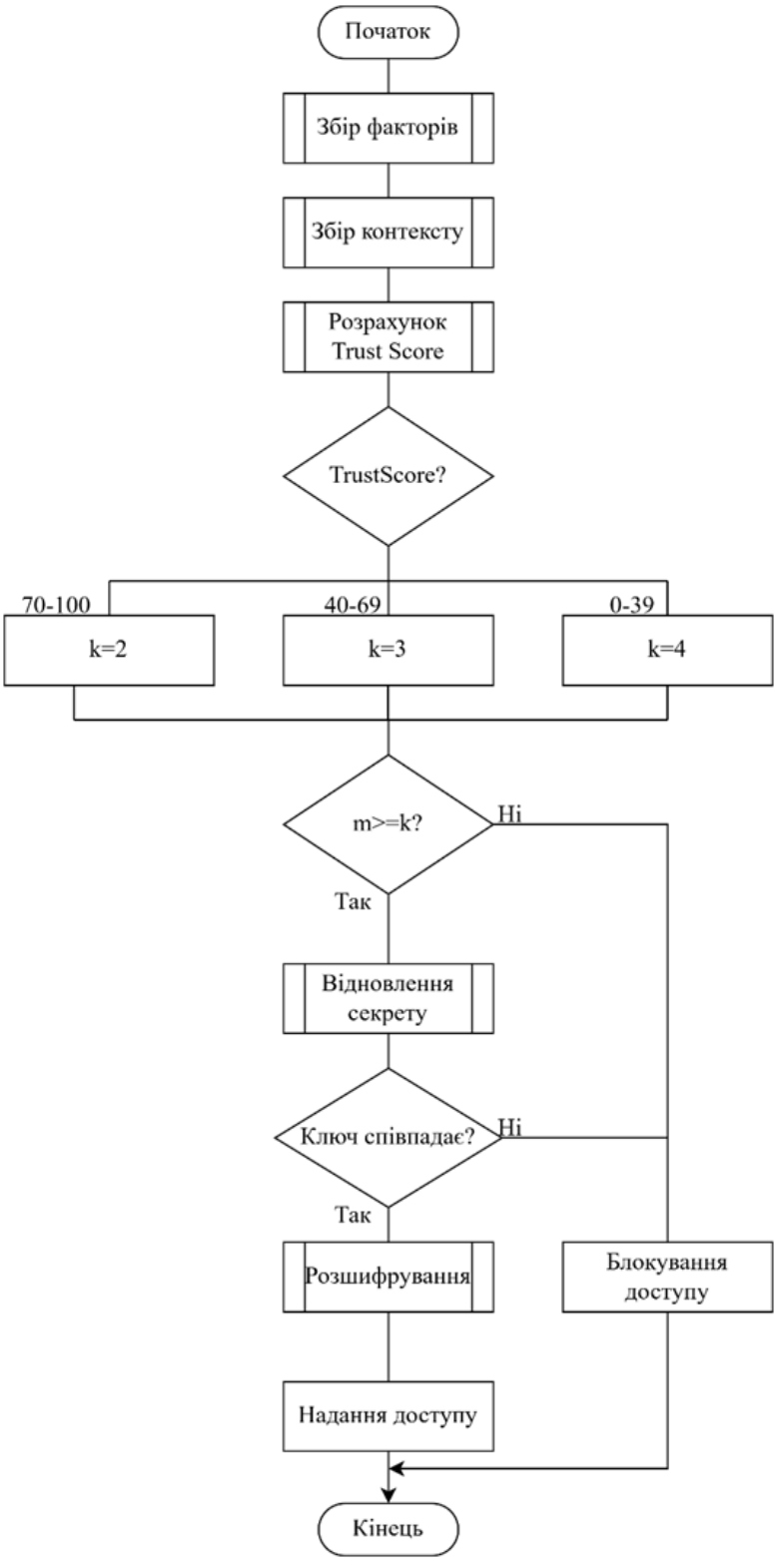
АЛГОРИТМ АДАПТИВНОГО ВИЗНАЧЕННЯ ПОРОГУ K



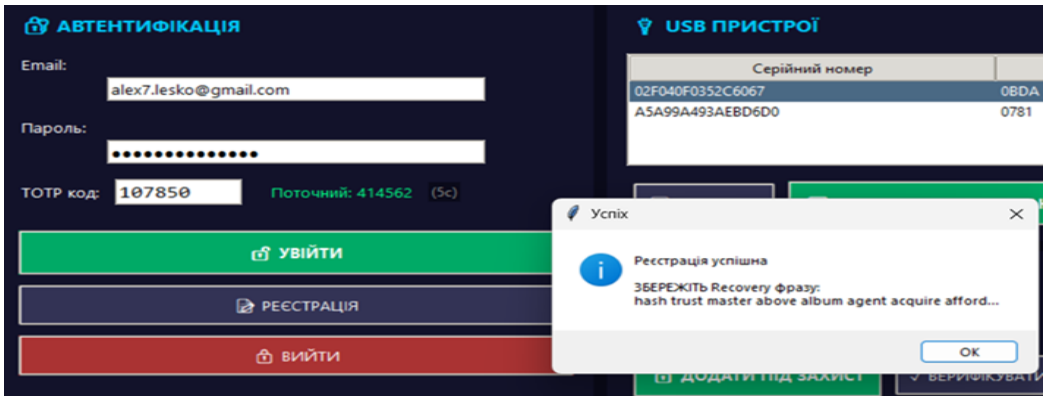
ДІАГРАМА СТАНІВ СИСТЕМИ АДАПТИВНОГО ВИЗНАЧЕННЯ ПОРОГУ K

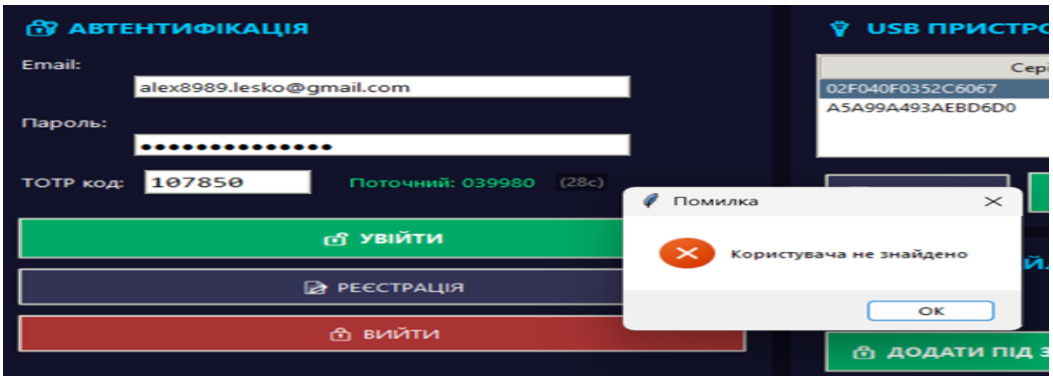


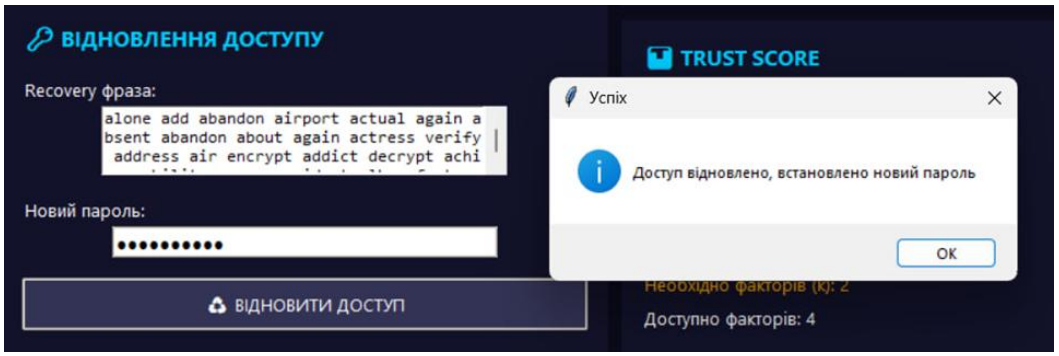
АЛГОРИТМ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧА

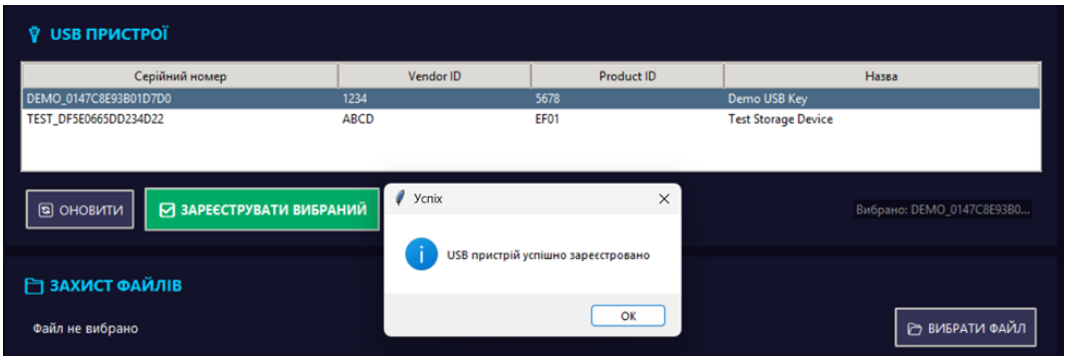


РЕЗУЛЬТАТИ ТЕСТУВАННЯ ІНТЕРФЕЙСУ









РЕЗУЛЬТАТИ МОДУЛЬНОГО ТЕСТУВАННЯ

№	Назва тесту	Компонент	Час, с	Результат
1	test_key_generation	CryptoModule	0.001	PASSED
2	test_pbkdf2_hashing	CryptoModule	0.089	PASSED
3	test_shamir_share_generation	ShamirSecretSharing	0.003	PASSED
4	test_shamir_reconstruction_k2	ShamirSecretSharing	0.002	PASSED
5	test_shamir_reconstruction_k3	ShamirSecretSharing	0.002	PASSED
6	test_insufficient_shares	ShamirSecretSharing	0.002	PASSED
7	test_totp_generation	TotpManager	0.001	PASSED
8	test_totp_window	TotpManager	0.001	PASSED
9	test_hwid_collection	HwidManager	0.015	PASSED
10	test_usb_scanning	UsbDeviceManager	0.008	PASSED
11	test_trust_score_normal	TrustScoreCalculator	0.001	PASSED
12	test_trust_score_anomaly	TrustScoreCalculator	0.001	PASSED
13	test_trust_score_risky	TrustScoreCalculator	0.001	PASSED
14	test_missing_usb	ProtectionController	0.001	PASSED
15	test_corrupted_shares	ShamirSecretSharing	0.002	PASSED
16	test_wrong_totp	TotpManager	0.001	PASSED
17	test_hwid_spoofing	HwidManager	0.001	PASSED
18	test_database_operations	DatabaseManager	0.089	PASSED

ПОКАЗНИКИ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

Критерії	Бали		
	Лебідь С.В.	Чухраєва О.В.	Стороженко А.С.
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	4	4	3
3. Ринкові переваги (ціна продукту)	4	4	3
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	4	4	4
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	2	2	2
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	2	3	2
10. Практична здійсненність (необхідність нових матеріалів)	3	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	42	44	41
Середньоарифметична сума балів $СБ_c$	42,3		