Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
«Метод і засіб захисту програмного засобу за рахунок прив'язки до оперативної пам'яті»

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

Керівник: к. т. н., доцент каф. ЗІ
_____Володимир ГАРНАГА
«_19_» _грудня_ 2025 р.

Опонент: д.т.н., проф., зав. каф. ПЗ
_____Олександр РОМАНЮК
«_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

Олександру СОКОЛЮКУ

1. Тема роботи: «Метод і засіб захисту програмного засобу за рахунок прив'язки до оперативної пам'яті»
керівник роботи: Володимир ГАРНАГА, к. т. н., доцент кафедри ЗІ, затверджені наказом ректора ВНТУ від 24 вересня 2025 року №313.
2. Строк подання студентом роботи 19 грудня 2025 р.
3. Вихідні дані до роботи:
   - Метод захисту програмного забезпечення – апаратно-програмна прив'язка до оперативної пам'яті з використанням TPM.
   - Алгоритм гешування – SHA-256.
   - Алгоритм шифрування – Serpent.
   - Джерело унікальних апаратних параметрів – структура та часові характеристики оперативної пам'яті (RAM).
   - Модуль підтвердження автентичності середовища – TPM.
   - Мова реалізації програмного засобу – Java (JDK 17).
   - Операційна система – Windows 10 / 11.
4. Зміст текстової частини: Вступ. 1. Аналіз інформаційних ресурсів. 2. Розробка методу та засобу захисту програмного забезпечення. 3. Експериментальні дослідження. 4. Економічна частина Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: архітектура апаратно-залежного контролю автентичності середовища виконання (TPM, Secure Boot, SGX), Структура оперативної пам'яті та основні параметри, що можуть бути використані для ідентифікації пристрою, Механізм перевірки цілісності системи з використанням TPM і PCR-регістрів, Механізм перевірки цілісності системи з використанням TPM і PCR-регістрів,

Загальна архітектура системи захисту програмного забезпечення. Загальний алгоритм методу прив'язки програмного забезпечення до оперативної пам'яті.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Володимир ГАРНАГА, к.т.н., доц. каф.ЗІ	25.09.25	10.12.25
2	Володимир ГАРНАГА, к.т.н., доц. каф.ЗІ	25.09.25	10.11.25
3	Володимир ГАРНАГА, к.т.н., доц. каф.ЗІ	25.09.25	10.12.25
4	Олександр ЛЕСЬКО, зав. каф. ЕПВМ, к. е. н., проф.	10.11.25	18.11.25

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 - 07.10.2025	
3	Науково-технічне обгрунтування	11.10.2025 - 22.10.2025	
4	Розробка технічного проєкту системи захисту	23.10.2025 - 26.10.2025	
5	Розробка математичної моделі та методу апаратної прив'язки	27.10.2025 - 02.11.2025	
6	Розробка криптографічних алгоритмів і механізму SecureLoader	03.11.2025 - 10.11.2025	
7	Експериментальні дослідження та аналіз результатів	10.11.2025 - 17.11.2025	
8	Розробка розділу економічного обгрунтування доцільності розробки	18.11.2025 - 22.11.2025	
9	Оформлення пояснювальної записки	23.11.2025 - 29.11.2025	
10	Перевірка магістерської роботи на наявність текстових запозичень	29.11.2025 - 11.12.2025	
11	Попередній захист та доопрацювання МКР	01.12.2025 - 15.12.2025	
12	Представлення МКР до захисту, рецензування	16.12.2025 - 19.12.2025	
13	Захист МКР	19.12.2025 - 23.12.2025	

Студент _____ Олександр СОКОЛЮК

Керівник роботи _____ Володимир ГАРНАГА

АНОТАЦІЯ

УДК 681.324.6

Соколюк О. М. Метод і засіб захисту програмного забезпечення за рахунок прив’язки до оперативної пам’яті.

Магістерська кваліфікаційна робота зі спеціальності 125 – Кібербезпека та захист інформації, освітня програма – Безпека інформаційних і комунікаційних систем. Вінниця: ВНТУ, 2025. с.

Укр. мовою. Бібліогр.: назв; рис.: ; табл.: .

Магістерська кваліфікаційна робота присвячена розробці методу та програмного засобу захисту програмного забезпечення шляхом прив’язки до параметрів оперативної пам’яті, Trusted Platform Module та криптографічно захищеного контейнера. Проведено науково-дослідне та техніко-економічне обґрунтування доцільності впровадження розробки. У роботі виконано аналіз сучасних методів протидії клонуванню й несанкціонованому модифікуванню ПЗ, обґрунтовано вибір RAM-профілювання, DeviceHash та механізму SecureLoader. Розроблено математичну модель прив’язки ПЗ до динамічних апаратних параметрів. Створено програмний засіб, який дозволив експериментально підтвердити стійкість розробленого методу до клонування системи, заміни модулів оперативної пам’яті та модифікації захищеного контейнера.

Ілюстративна частина містить схемні діаграми, структурні моделі, результати тестових запусків програмного засобу та графічні матеріали експериментальних досліджень.

В економічному розділі оцінено витрати на розробку та визначено економічну ефективність комерціалізації.

Ключові слова: захист програмного забезпечення, обфускація, ram-профілювання, tpm, devicehash, криптоконтейнер, secureloader.

ABSTRACT

Sokoliuk O. M. Method and tool for software protection based on volatile memory binding.

Master's Thesis in specialty 125 – Cybersecurity and Information Protection, educational program – Security of Information and Communication Systems. Vinnytsia: Vinnytsia National Technical University, 2025. p.

In English. References: items; figures: ; tables: .

The Master's Thesis is devoted to the development of a method and software tool for protecting software by binding it to the parameters of the system's volatile memory, Trusted Platform Module, and a cryptographically secured container. A scientific, technical, and economic justification for the feasibility of the proposed approach is provided. The work includes an analysis of existing mechanisms for countering cloning and unauthorized modification of software, substantiating the use of RAM profiling, DeviceHash formation, and the SecureLoader mechanism. A mathematical model of binding software to dynamic hardware characteristics is developed.

A software tool implementing the proposed method was created, enabling experimental verification of its resistance to system cloning, RAM module replacement, and cryptographic container modification. The illustrative materials include structural diagrams, workflow schemes, results of experimental launches, and visual data representing the outcomes of the performed tests.

The economic section evaluates the cost of development and determines the economic efficiency of potential commercialization.

Keywords: software protection, obfuscation, ram profiling, tpm, devicehash, crypto container, secureloader.

ЗМІСТ

ВСТУП

У сучасних умовах розвитку цифрових технологій проблема захисту програмного забезпечення від несанкціонованого копіювання, модифікації та запуску у неавторизованому середовищі залишається однією з найактуальніших у галузі кібербезпеки. Значна частина комерційних і промислових рішень створюється з використанням складних алгоритмів та великих обсягів даних, тому витік або компрометація таких програм може призвести до суттєвих економічних та інформаційних втрат. Традиційні методи захисту, засновані лише на перевірці серійних номерів пристроїв, ліцензійних ключах або мережевій аутентифікації, у сучасних умовах розвитку технологій реверс-інжинірингу та віртуалізації втрачають ефективність.

Значний інтерес викликає використання апаратних характеристик системи для створення унікального середовища виконання програмного забезпечення. Зокрема, параметри оперативної пам’яті, її структура та серійні номери модулів можуть бути використані як елемент апаратного профілю пристрою. Це дає змогу формувати стійку до підміни та емуляції ідентифікацію, що може бути використана як основа для захисту програм. Такі підходи доповнюються застосуванням модуля довіреної платформи (Trusted Platform Module, TPM), який дозволяє створювати сертифікати автентичності та апаратно забезпечувати криптографічні операції [1].

Проблематика створення захищених програмних систем активно досліджується вітчизняними та закордонними науковцями. Вагомий внесок у розвиток методів захисту інформації зробили Корченко О. Г., Куперштейн Л. М., Рудницький В. М., Лужецький В. А., які займаються питаннями безпеки програмних засобів та криптографічного захисту в Україні. Серед зарубіжних науковців особливу увагу приділяють дослідженням у цій сфері Брюс Шнайєр (B. Schneier, США), Росс Андерсон (R. Anderson, Велика Британія), Мелісса Дворкін (M. Dworkin, NIST, США). Дослідження проводяться у провідних наукових установах, таких як Massachusetts Institute of Technology (США), Fraunhofer Institute for Secure Information Technology (Німеччина), Інститут

інформаційних технологій НАН України (Київ, Україна), Вінницький національний технічний університет (Вінниця, Україна). Це свідчить про високу актуальність тематики як для міжнародної наукової спільноти, так і для національної кібербезпеки України, де захист власних програмних продуктів є стратегічним завданням [2].

Об’єктом дослідження є процес захисту програмного забезпечення від несанкціонованого копіювання, модифікації та виконання у неавторизованому середовищі.

Предметом дослідження є методи, моделі та програмні засоби прив’язки програмного забезпечення до апаратних параметрів, зокрема до оперативної пам’яті, з використанням криптографічних алгоритмів.

Метою магістерської кваліфікаційної роботи є підвищення рівня захисту програмного забезпечення шляхом розроблення методу та програмного засобу, що забезпечує його прив’язку до характеристик оперативної пам’яті комп’ютера і параметрів модуля TPM. Досягнення цієї мети дозволяє забезпечити підвищення стійкості до підміни середовища виконання, унеможливлення запуску у віртуальному середовищі та запобігання копіюванню програмних модулів.

Для досягнення поставленої мети необхідно:

– проаналізувати сучасні методи і засоби захисту програмного забезпечення від копіювання та модифікації;

– дослідити можливості використання апаратних параметрів оперативної пам’яті та TPM-модуля для формування унікального криптографічного профілю системи;

– розробити метод побудови криптографічного ключа на основі параметрів оперативної пам’яті;

– створити програмний засіб, що реалізує захист із використанням алгоритму Serpent та гешування SHA-256;

– провести експериментальну перевірку працездатності та стійкості розробленого методу;

– оцінити ефективність запропонованого підходу у порівнянні з існуючими засобами захисту.

Наукова новизна магістерської кваліфікаційної роботи полягає в тому, що:

– удосконалено метод формування криптографічного профілю середовища виконання, що забезпечує неможливість запуску програмного засобу у віртуальному або підміненому апаратному середовищі;

– розроблено програмний засіб, який поєднує апаратну прив’язку, криптографічне шифрування та перевірку цілісності даних, що дозволяє підвищити рівень захисту від несанкціонованого копіювання та аналізу.

1 АНАЛІЗ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

У сучасних умовах розвитку інформаційних технологій питання забезпечення надійного захисту програмного забезпечення набуває особливої актуальності. З кожним роком збільшується кількість випадків несанкціонованого копіювання, модифікації, підміни або запуску програмних продуктів у неавторизованому середовищі. Такі дії не лише завдають значних економічних збитків розробникам, але й створюють серйозні ризики для безпеки критичних інформаційних систем.

Одним із головних завдань сучасної кібербезпеки є створення методів, здатних забезпечити стійкий захист програмного забезпечення від реверс-інжинірингу та нелегального розповсюдження. Для цього необхідно не тільки використовувати криптографічні алгоритми, але й поєднувати їх з апаратними механізмами перевірки цілісності та автентичності середовища виконання. Саме така інтеграція дозволяє створити захищене середовище, у якому програмний код може виконуватись лише на визначеній платформі.

Сучасні підходи до захисту програмного забезпечення, як правило, базуються на методах ліцензування, обфускації, цифрових підписів, використанні апаратних ідентифікаторів та криптографічних протоколів. Проте більшість із них не враховують динамічних характеристик комп’ютерної системи, зокрема оперативної пам’яті, яка є одним із найменш досліджених, але потенційно найефективніших джерел унікальних ознак пристрою [3].

Оперативна пам’ять, як складова апаратного середовища, має фізичні та часові характеристики, що залежать від конкретного обладнання. Це відкриває можливість використання параметрів пам’яті для формування унікального профілю комп’ютера, який може бути використаний як база для прив’язки програмного забезпечення. Поєднання таких апаратних ознак із криптографічними методами створює новий рівень захисту від підміни середовища виконання, віртуалізації або спроб копіювання програми [4].

У даному розділі проведено аналіз сучасного стану проблеми захисту

програмного забезпечення, розглянуто відомі методи і підходи до забезпечення безпеки, виконано порівняння їхніх переваг і недоліків, а також досліджено можливості застосування апаратно-залежних характеристик, зокрема параметрів оперативної пам’яті, для підвищення рівня захисту програмних систем. Результатом аналізу є постановка завдання магістерської кваліфікаційної роботи, спрямованої на розроблення методу та програмного засобу захисту програмного забезпечення за рахунок прив’язки до оперативної пам’яті комп’ютера.

1.1 Сучасний стан проблеми захисту програмного забезпечення

Сучасна індустрія програмного забезпечення функціонує в умовах, де інтелектуальна власність, втілена у коді, є ключовим і водночас надзвичайно вразливим активом. У зв'язку з цим, захист програмних засобів перетворився з простої перевірки ліцензії на комплексну, багаторівневу дисципліну. Спостерігається чіткий зсув від пасивних методів захисту до проактивних. Ключовою тенденцією є перехід до моделі "Нульової довіри" (Zero Trust), яка відкидає застаріле припущення про те, що середовище виконання програми є надійним. Натомість сучасні системи вимагають постійної верифікації не лише користувача, але й самого апаратного та програмного оточення, в якому працює код [5].

Цей підхід вимагає застосування багаторівневого захисту (Defense in Depth), оскільки жоден окремий метод не може гарантувати повної безпеки. Ефективні рішення комбінують різні шари оборони. Наприклад, для ускладнення аналізу логіки програми застосовується обфускація, яка ускладнює розуміння декомпільованого коду. Це, у свою чергу, доповнюється шифруванням критичних компонентів програми, які розшифровуються і завантажуються безпосередньо в оперативну пам'ять лише в момент виконання, що захищає від статичного аналізу файлів [6]. Усі ці програмні методи підсилюються фундаментальним шаром – апаратною прив'язкою, яка є основою

даного дослідження.

Ці тенденції є прямою відповіддю на спектр загроз, що постійно еволюціонують. Найбільш фундаментальною загрозою є реверс-інжиніринг, тобто процес декомпіляції та аналізу програми для викрадення алгоритмів, пошуку вразливостей або підготовки до модифікації. Навіть обфускація чи приховування системних команд (наприклад, через зашифровані рядки) не зупиняють зловмисника, а лише виграють час. Отримавши уявлення про логіку, зловмисник переходить до модифікації (крекінгу), змінюючи код програми – наприклад, обходячи перевірку ліцензії. Для боротьби з цим застосовують механізми контролю цілісності, такі як перевірка цифрових підписів або звірка хеш-сум критичних блоків даних перед їх використанням [6].

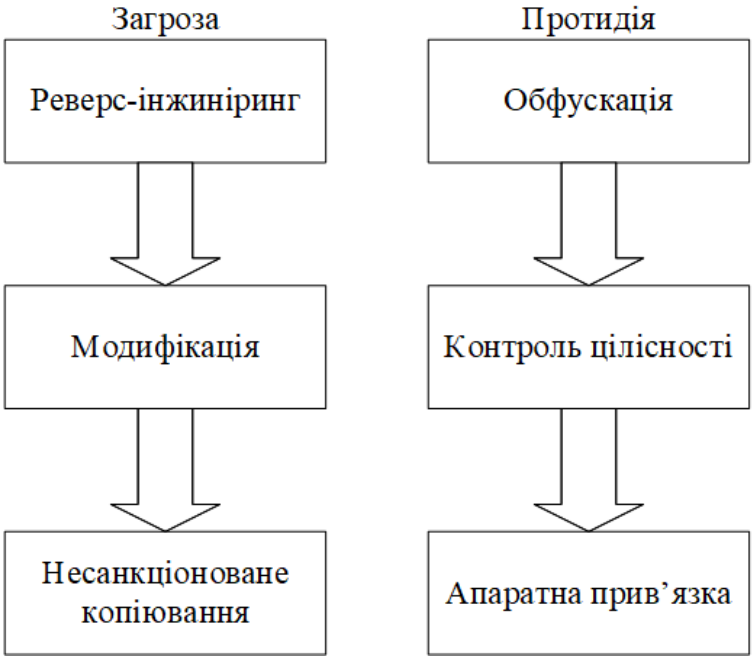


Рисунок 1.1 – Модель загроз сучасному ПЗ та рівні захисту

Найпоширенішою загрозою залишається несанкціоноване копіювання, коли програма просто переноситься на інший комп'ютер. Саме для боротьби з цим історично виникли методи апаратної прив'язки. Однак з часом з'явилася більш витончена загроза – підміна середовища виконання. У цьому сценарії зловмисник не копіює програму, а змушує її "думати", що вона працює на

легітимній машині. Це досягається шляхом емуляції або "підробки" (спуфінгу) апаратних ідентифікаторів, таких як MAC-адреса мережевої карти або серійний номер жорсткого диска. Саме ця загроза знецінює прості методи апаратної прив'язки і змушує шукати більш надійні "якорі" [7].

У відповідь на ці виклики роль апаратної прив'язки кардинально змінилася. Вона еволюціонувала від простого зчитування статичних ID до використання надійних апаратних коренів довіри (Hardware Roots of Trust). Сучасні рішення все частіше покладаються на спеціалізовані криптографічні компоненти, такі як Trusted Platform Module (TPM). Головна перевага TPM полягає в тому, що він може не просто повідомити свій ідентифікатор, але й криптографічно довести свою автентичність. Замість пасивного зчитування ID, програма може провести активну атестацію: згенерувати випадковий запит (nonce) і вимагати від TPM підписати його своїм унікальним, вбудованим приватним ключем. Підробити таку криптографічну операцію практично неможливо [1].

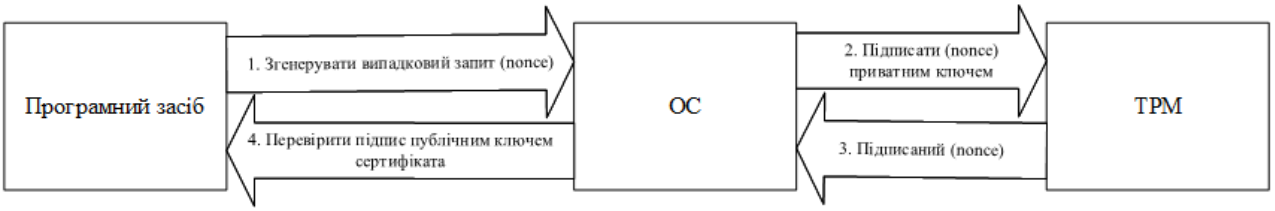


Рисунок 1.2 – Принцип роботи активної атестації через TPM

Для посилення захисту від спуфінгу та підвищення надійності системи (адже TPM є не на всіх пристроях), сучасні методи використовують композитні ідентифікатори. Замість того, щоб покладатися на один компонент, система збирає унікальні характеристики з кількох джерел і об'єднує їх в єдиний "апаратний відбиток" (хеш). Цей підхід є центральним для даної роботи, оскільки він дозволяє поєднати переваги різних компонентів. Зокрема, використання серійного номера оперативної пам'яті (RAM) є вкрай ефективним, оскільки цей ідентифікатор є фізично унікальним для модуля пам'яті, важкодоступним для програмної підробки (вимагає специфічних

системних викликів) і водночас не залежить від інших компонентів, що легко змінюються (як HDD або SSD) [8].

Таблиця 1.1 – Аналіз джерел ідентифікаторів для апаратної прив'язки [9]

Джерело Ідентифікатора (ID)	Стійкість до Підробки (Спуфінгу)	Стабільність (після заміни компонента)	Обґрунтування
Мережева карта (MAC-адреса)	Низька	Низька	Легко змінюється програмно (MAC spoofing), особливо у віртуальних машинах. Не є унікальною прив'язкою до фізичного пристрою.
Жорсткий диск/SSD (Serial Number)	Середня	Низька	Можна емулювати або змінити за допомогою спеціалізованого програмного забезпечення. Змінюється при штатному апгрейді системи.
Центральний процесор (CPU ID)	Висока	Дуже висока	Ідентифікатор є вбудованим і фізично незмінним. Недолік: потребує специфічних інструкцій (наприклад, CPUID) і не забезпечує криптографічної довіри.
Оперативна пам'ять (RAM Serial Number)	Висока	Середня	Доступ до серійного номера через SPD-чіп вимагає складних системних викликів (WMI). Важко підробити у віртуальному середовищі. Змінюється при апгрейді RAM.
Довірений Платформний Модуль (TPM Attestation Key)	Дуже висока	Висока	Криптографічний корінь довіри. Забезпечує динамічну атестацію (доказ володіння ключем), що неможливо підробити. Зазвичай прив'язаний до материнської плати.

Таким чином, комбінація статичної, унікальної ознаки (серійний номер RAM) та динамічного, криптографічно захищеного доказу (атестація TPM) дозволяє побудувати надійний, багатофакторний апаратний профіль, що

ефективно протистоїть як простому копіюванню, так і складним атакам з підміною середовища.

1.2 Аналіз відомих методів захисту програмного забезпечення від несанкціонованого копіювання

Проблема несанкціонованого копіювання програмного забезпечення виникла практично одночасно із його поширенням у комерційній сфері. Незважаючи на багаторічний досвід досліджень у цій галузі, на сьогодні не існує універсального рішення, яке б повністю гарантувало захист програмного коду від копіювання чи аналізу. Усі сучасні методи мають свої переваги, недоліки та сфери застосування, а їхня ефективність значною мірою залежить від умов експлуатації програмного продукту.

Одними з найпоширеніших програмних способів захисту є ліцензійні ключі та серійні номери. Сутність цих методів полягає у перевірці автентичності користувача або пристрою за допомогою спеціального коду, який генерується на основі унікальних параметрів системи чи записаний у базі даних сервера активації. Такий підхід реалізовано в системах Microsoft Volume Licensing, Adobe Creative Cloud тощо. Перевагою є простота реалізації та можливість централізованого контролю за ліцензіями. Недоліком є низька стійкість до підміни чи підбору ключів, а також наявність великої кількості неофіційних генераторів (keygen), які дозволяють обходити перевірку [9].

Іншим класом методів є контроль цілісності програмного коду. Він базується на порівнянні контрольних хеш-значень (MD5, SHA-256 тощо) оригінального файлу із поточним станом при виконанні програми. Якщо виявлено невідповідність, програма припиняє роботу або блокує доступ до своїх функцій. Прикладом є реалізація цього підходу в антивірусних системах Kaspersky Endpoint Security та ESET NOD32. Перевага методу полягає у здатності виявляти модифікації коду, проте він не запобігає його копіюванню або відтворенню у новому середовищі [10].

Широко застосовується також обфускація програмного коду – перетворення структури програми у форму, складну для аналізу людиною або дизасемблером. Обфускатори замінюють зрозумілі імена змінних і функцій на випадкові, змінюють структуру умов, вставляють “порожні” гілки коду або шифрують частини байт-коду. Відомі інструменти, такі як ProGuard, Themida, VMProtect, поєднують обфускацію з додатковими антидебаг-засобами. Перевагою є ускладнення реверс-інжинірингу, проте недоліком залишається значне зниження продуктивності та можливість автоматичного відновлення структури коду за допомогою спеціалізованих декомпіляторів [11].

До криптографічних методів належить використання цифрових підписів. Програмне забезпечення підписується приватним ключем розробника, а під час інсталяції або запуску перевіряється відповідним відкритим ключем чи сертифікатом. Прикладом є Microsoft Authenticode або Linux GPG-сигнатури пакетів. Цей підхід забезпечує автентичність джерела та цілісність файлу, однак не захищає від копіювання самого файлу або запуску його на іншому пристрої, якщо підпис залишається дійсним.

Серед сучасних технологій особливу роль відіграють апаратно-програмні рішення, які поєднують перевірку апаратних параметрів із криптографічними операціями. Це можуть бути USB-токени, модулі TPM, апаратні ключі типу HASP, eToken, або захист із використанням унікальних параметрів процесора. Такі методи суттєво підвищують рівень безпеки, оскільки передбачають фізичну прив’язку до конкретного пристрою. Проте їх впровадження потребує додаткових витрат і ускладнює розповсюдження програмного забезпечення [12].

Для порівняння ефективності розглянутих методів у таблиці 1.2 наведено їх основні характеристики.

Таблиця 1.2 – Порівняння відомих методів захисту програмного забезпечення від несанкціонованого копіювання

№	Метод захисту	Переваги	Недоліки	Стійкість до атак
1	Ліцензійні ключі, серійні номери	Простота реалізації, централізований контроль	Легко підробити, існують генератори ключів	Низька
2	Контроль цілісності	Виявляє модифікації коду	Не запобігає копіюванню	Середня
3	Обфускація коду	Ускладнює реверс-інжиніринг	Знижує швидкодію, можлива декомпіляція	Середня
4	Цифрові підписи	Гарантують автентичність і цілісність	Не заважають копіюванню файлу	Середня
5	Апаратно-програмна прив’язка	Висока надійність, стійкість до підміни	Висока вартість, складність реалізації	Висока

На основі проведеного аналізу можна зробити висновок, що для створення ефективної системи захисту програмного забезпечення доцільно комбінувати кілька методів: програмні (обфускація, контроль цілісності) та апаратно-криптографічні (прив’язка до TPM, до характеристик оперативної пам’яті). Такий підхід дозволяє досягнути високої стійкості до копіювання, модифікації та підміни середовища виконання.

1.3 Апаратно-залежні методи ідентифікації програмного середовища

Одним із найефективніших напрямів підвищення стійкості систем захисту програмного забезпечення є використання апаратно-залежних методів ідентифікації середовища, у якому виконується програма. Сутність таких методів полягає у формуванні унікального «відбитка» (hardware fingerprint), що базується на фізичних або логічних параметрах комп’ютерного обладнання.

Програмне забезпечення, яке перевіряє ці параметри під час запуску, може визначати, чи виконується воно на авторизованому пристрої, і блокувати роботу у разі виявлення невідповідності.

Найпростішим способом апаратної ідентифікації є перевірка серійних номерів компонентів системи – центрального процесора (CPU ID), жорсткого диска (HDD Serial), материнської плати або мережевого інтерфейсу (MAC-адреса). Такі параметри отримуються через стандартні системні інтерфейси, зокрема BIOS, SMBIOS або WMI (Windows Management Instrumentation).

Цей підхід використовується у великій кількості ліцензійних систем, наприклад у продуктах Autodesk, ABBYY, Microsoft, де активаційний код формується на основі комбінації декількох серійних номерів пристроїв.

Основною перевагою цього методу є простота реалізації та мінімальні вимоги до апаратури. Недоліком є низький рівень стійкості, оскільки більшість зазначених параметрів можна змінити або підмінити програмно. Існують утиліти, які дозволяють підробити MAC-адресу, змінити серійний номер жорсткого диска або навіть згенерувати довільний ідентифікатор процесора. Таким чином, у сучасних умовах такий метод можна розглядати лише як допоміжний засіб захисту [13].

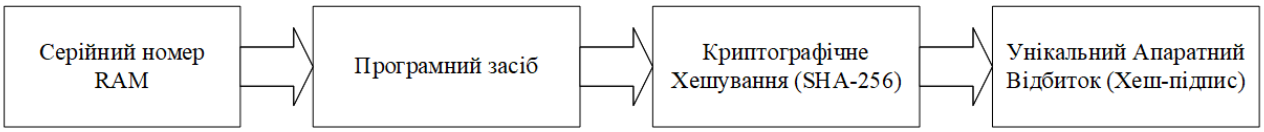


Рисунок 1.3 – Схема формування унікального апаратного відбитка на основі серійних номерів пристроїв

Сучасний рівень розвитку засобів апаратного захисту дозволяє використовувати спеціалізовані модулі безпеки, які забезпечують апаратне збереження ключів і підтвердження автентичності платформи. До таких технологій належать Trusted Platform Module (TPM), Secure Boot і Intel Software Guard Extensions (SGX).

Trusted Platform Module (TPM) – це мікроконтролер, який реалізує

криптографічні функції на апаратному рівні: генерацію, зберігання та використання криптографічних ключів, формування сертифікатів автентичності, а також обчислення геш-значень компонентів системи. TPM містить у собі так звані PCR-регістри (Platform Configuration Registers), у яких фіксується стан системи під час її завантаження. Програмний засіб може використовувати ці значення для перевірки автентичності середовища виконання. Перевага TPM полягає у тому, що він забезпечує апаратну немодифікованість ключів, а отже – високий рівень довіри до перевірених даних. Недоліком є складність інтеграції в програмні продукти, що не мають прямого доступу до низькорівневих API [14].

Secure Boot – це технологія, реалізована в BIOS/UEFI, що гарантує завантаження лише підписаних цифровим сертифікатом компонентів операційної системи. Вона запобігає запуску модифікованих або шкідливих драйверів і дозволяє перевіряти цілісність системного середовища перед стартом програмного забезпечення. Хоча Secure Boot не здійснює ідентифікацію самого пристрою, він забезпечує цілісність і довіру до процесу завантаження, що є необхідною умовою для подальшої перевірки програм[15].

Intel Software Guard Extensions (SGX) – це технологія апаратної ізоляції, яка дозволяє створювати так звані “enclave” – захищені області пам’яті, доступ до яких має лише довірений код. Використання SGX забезпечує виконання критичних обчислень у середовищі, захищеному від зовнішніх втручань. Такий підхід дозволяє створювати модулі програмного захисту, стійкі до спроб аналізу, модифікації або підміни даних під час виконання. Недоліком SGX є обмежена сумісність із програмними платформами та необхідність спеціальної підтримки з боку процесора [16].

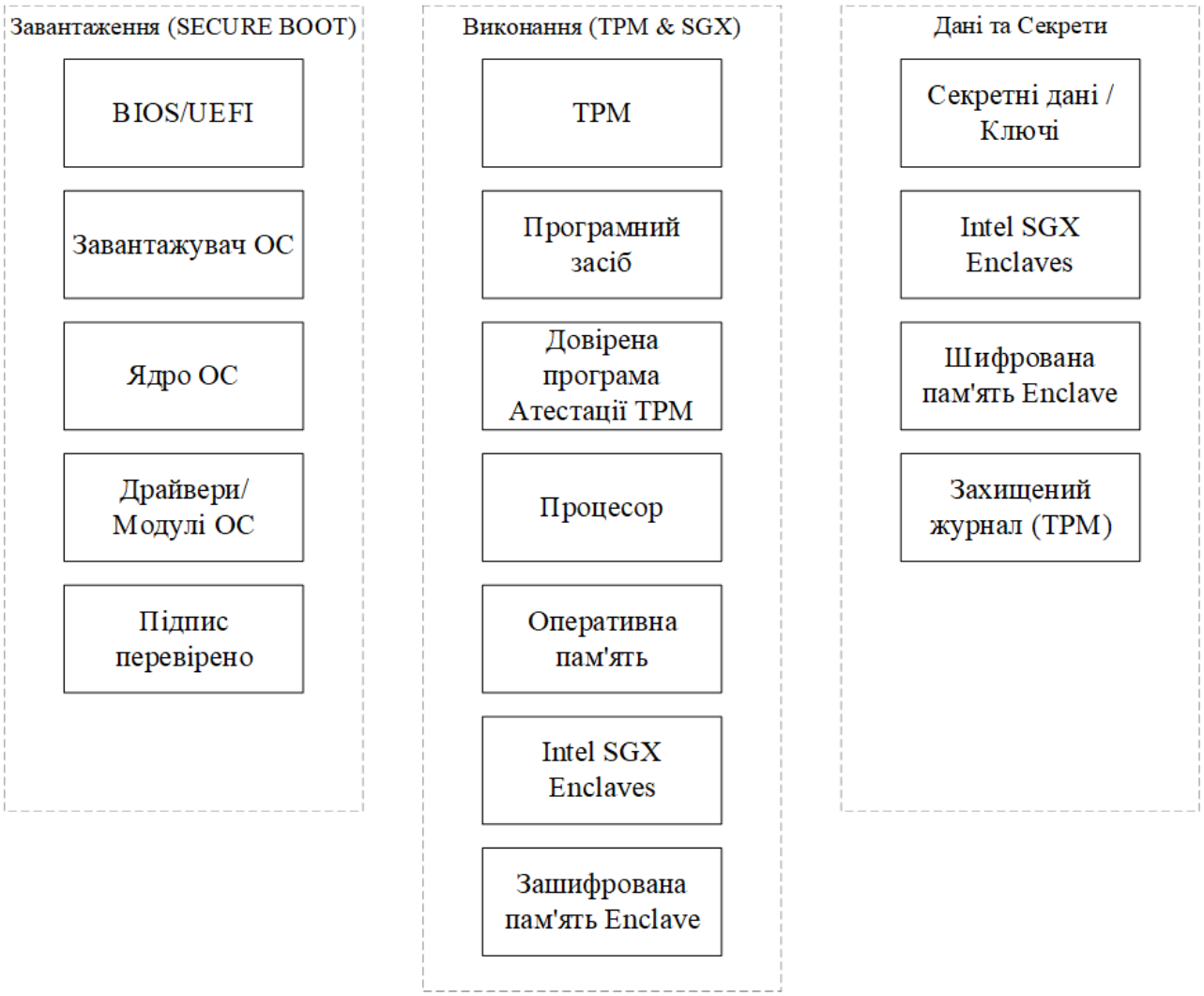


Рисунок 1.4 – Архітектура апаратно-залежного контролю автентичності середовища виконання (TPM, Secure Boot, SGX)

Апаратно-залежні методи забезпечують вищий рівень безпеки порівняно з виключно програмними засобами, оскільки базуються на фізичних параметрах пристрою, які складно або неможливо підробити. Водночас вони не позбавлені вразливостей. До основних належать:

- можливість емуляції апаратних параметрів у віртуальних середовищах;
- залежність від конкретної архітектури пристрою (Intel, AMD, ARM);
- складність реалізації універсальних API для взаємодії програм із апаратними компонентами;
- висока вартість впровадження при масовому використанні.

Для підвищення стійкості системи ідентифікації доцільно поєднувати

декілька апаратних джерел, наприклад TPM та оперативну пам’ять, з додатковими криптографічними перевірками. Саме такий підхід дозволяє створити унікальний апаратно-програмний профіль середовища, що практично унеможливлює запуск програмного забезпечення на підмінених або віртуалізованих системах.

Таблиця 1.3 – Порівняльна характеристика апаратно-залежних методів ідентифікації програмного середовища

№	Метод	Переваги	Недоліки	Рівень захисту
1	Серійні номери пристроїв (CPU, HDD, MAC)	Простота реалізації	Легко підмінюються або емуються	Низький
2	TPM	Апаратна цілісність, захист ключів	Складна інтеграція, потребує підтримки ОС	Високий
3	Secure Boot	Гарантує цілісність процесу завантаження	Не ідентифікує пристрій напряму	Середній
4	Intel SGX	Захищене середовище виконання, стійкість до аналізу	Обмежена сумісність, залежність від CPU	Високий

Таким чином, аналіз показує, що апаратно-залежні методи ідентифікації є ключовим напрямом розвитку сучасних систем захисту програмного забезпечення. Їхнє подальше вдосконалення полягає у пошуку нових джерел унікальних апаратних ознак, зокрема параметрів оперативної пам’яті, які

можуть стати основою для формування криптографічно стійкого профілю середовища виконання. Саме ця ідея покладена в основу методу, розробленого у даній магістерській роботі.

1.4 Роль оперативної пам’яті у забезпеченні безпеки програмного забезпечення

Оперативна пам’ять (RAM) є одним із ключових апаратних компонентів комп’ютерної системи, що забезпечує тимчасове зберігання даних під час виконання програм. На відміну від постійних накопичувачів, оперативна пам’ять характеризується високою швидкістю доступу, динамічністю структури та залежністю від фізичних параметрів апаратури. Ці властивості роблять її не лише важливою складовою продуктивності системи, а й перспективним елементом у побудові засобів захисту програмного забезпечення.

Структура оперативної пам’яті визначається фізичною організацією модулів DRAM або SDRAM, топологією банків, кількістю рядків і стовпців комірок, а також характеристиками контролера пам’яті. Кожен модуль RAM має унікальний серійний номер, ідентифікатор виробника та параметри таймінгів (CL, RAS, CAS, tRCD, tRP), які фіксуються у мікросхемі SPD (Serial Presence Detect). Ці дані можна зчитувати через стандартні інтерфейси SMBus або BIOS.

Сукупність таких параметрів формує індивідуальний апаратний профіль модуля пам’яті. На практиці комбінація SPD-даних декількох модулів може використовуватись для створення криптографічного ідентифікатора системи, подібно до відбитка пристрою. При цьому навіть заміна одного модуля призводить до зміни контрольної суми, що забезпечує високу чутливість до несанкціонованих модифікацій апаратного середовища [17].

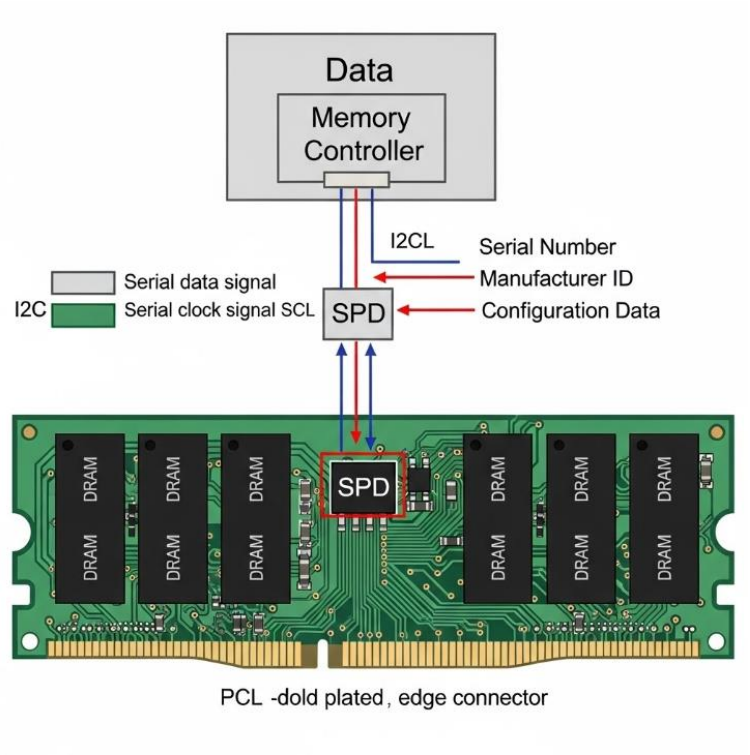


Рисунок 1.5 – Структура оперативної пам’яті та основні параметри, що можуть бути використані для ідентифікації пристрою.

Параметри оперативної пам’яті поділяються на статичні та динамічні. До статичних характеристик належать серійний номер, обсяг, частота, виробник та архітектура модуля. Вони залишаються незмінними впродовж усього часу експлуатації. Саме статичні параметри можуть слугувати базовою частиною ідентифікаційного профілю системи.

Динамічні характеристики – це показники, що залежать від умов роботи системи: затримки доступу до комірок, тривалість циклу оновлення (refresh rate), часові флуктуації, температурна залежність або рівень шумів на шинах даних. Дослідження, проведені в лабораторіях Fraunhofer Institute for Secure Information Technology (Німеччина) та MIT Computer Science and Artificial Intelligence Laboratory (США), показали, що такі мікроскопічні варіації можуть використовуватись для побудови фізично неклонованих функцій (PUF – Physically Unclonable Function).

PUF-механізми дозволяють створювати унікальні криптографічні ключі на основі поведінки конкретного модуля пам’яті, яку неможливо повторити або

підмінити. Це забезпечує новий рівень автентифікації, де навіть повна копія програми не зможе функціонувати на іншому апаратному середовищі [18].

Поєднання статичних і динамічних параметрів оперативної пам’яті дозволяє сформувати апаратний профіль пристрою, що може бути використаний як база для прив’язки програмного забезпечення. Такий профіль може бути представлений у вигляді хешу, отриманого шляхом обчислення SHA-256 від набору SPD-даних і статистичних параметрів часу доступу до пам’яті.

Під час встановлення програмного продукту формується еталонний профіль системи, який зберігається у зашифрованому вигляді. При кожному наступному запуску програмне забезпечення повторно зчитує параметри пам’яті та порівнює отриманий хеш із збереженим. У разі виявлення розбіжності програма припиняє роботу, що забезпечує захист від перенесення або запуску в іншому середовищі.

Крім того, комбінація апаратного профілю оперативної пам’яті з даними модуля TPM або Secure Boot створює багаторівневу систему довіри, у якій кожен компонент підтверджує автентичність іншого. Такий підхід дозволяє підвищити стійкість до віртуалізації, апаратної емуляції та підміни серійних номерів.

Таким чином, оперативна пам’ять є не лише функціональним елементом комп’ютерної архітектури, а й потенційним джерелом унікальних апаратних ознак, що можуть використовуватись для побудови систем захисту програмного забезпечення. Завдяки комбінації статичних і динамічних характеристик можливо створити криптографічно стійкий профіль середовища виконання, який забезпечує прив’язку програмного продукту до конкретної фізичної платформи.

Саме цей принцип покладено в основу методу захисту програмного засобу за рахунок прив’язки до оперативної пам’яті, розробленого у межах даної магістерської роботи.

1.5 Аналіз криптографічних алгоритмів, придатних для побудови апаратно-прив’язаних систем захисту

Побудова ефективних систем захисту програмного забезпечення потребує використання надійних криптографічних алгоритмів, які забезпечують цілісність, конфіденційність і автентичність даних. Особливої уваги потребують ті алгоритми, що здатні інтегруватися з апаратними характеристиками пристрою, утворюючи криптографічно стійкий зв’язок між програмним кодом та фізичним середовищем його виконання.

У даному підрозділі проведено короткий аналіз сучасних криптографічних стандартів – AES, Serpent, SHA-256, RSA та ECC – з метою визначення доцільності їх застосування для реалізації методу прив’язки до оперативної пам’яті.

AES (Advanced Encryption Standard) – симетричний алгоритм блочного шифрування, прийнятий як міжнародний стандарт FIPS-197. Він забезпечує високу швидкодію та широке апаратне прискорення (особливо в процесорах Intel і AMD, які підтримують інструкції AES-NI). Однак його структура є фіксованою, що ускладнює гнучку модифікацію під специфічні умови, наприклад – під унікальні параметри оперативної пам’яті. Тому AES доцільно використовувати для шифрування великих обсягів даних, але не як основний механізм ідентифікації апаратного середовища [19].

Serpent – один із фіналістів конкурсу AES, який відрізняється підвищеною криптографічною стійкістю. Його структура включає 32 раунди замін та перестановок, що робить його більш захищеним від диференційного та лінійного криптоаналізу, хоча й повільнішим у виконанні. Головна перевага Serpent полягає у гнучкості побудови ключового простору та можливості адаптації під унікальні апаратні параметри. Це дозволяє використовувати Serpent як базовий механізм шифрування у системах, де ключ формується на основі характеристик оперативної пам’яті [20].

SHA-256 – алгоритм гешування, який формує 256-бітове геш-значення на

основі вхідних даних довільної довжини. Він не є оборотним і використовується для перевірки цілісності та побудови цифрових підписів. У системі апаратно-прив’язаного захисту SHA-256 може застосовуватись для формування унікального «профілю» пристрою на основі параметрів оперативної пам’яті та подальшої перевірки незмінності середовища виконання [11].

RSA (Rivest–Shamir–Adleman) – класичний асиметричний алгоритм, який базується на складності факторизації великих чисел. Його перевага – у можливості безпечного розподілу ключів та цифрового підпису, але висока обчислювальна складність робить RSA менш ефективним для інтеграції у швидкі процеси перевірки середовища. RSA зазвичай використовується для початкової автентифікації та обміну ключами, а не для безпосереднього шифрування або перевірки стану пам’яті [21].

ECC (Elliptic Curve Cryptography) – сучасний асиметричний підхід, що забезпечує аналогічний рівень безпеки при суттєво менших розмірах ключів у порівнянні з RSA. Алгоритми на основі еліптичних кривих часто застосовуються у вбудованих системах (TPM, SecureBoot) для цифрового підпису та підтвердження автентичності пристроїв. Проте ECC потребує складної математичної обробки та не має прямої прив’язки до фізичних параметрів пам’яті [22].

Таблиця 1.4 – Порівняння сучасних криптографічних алгоритмів за основними характеристиками

№	Алгоритм	Тип	Основна перевага	Недолік
1	AES	Симетричний	Висока швидкодія, апаратна підтримка	Складно прив’язати до фізичних параметрів
2	Serpent	Симетричний	Висока стійкість, гнучка структура ключа	Повільніше виконання

Продовження таблиці 1.4

№	Алгоритм	Тип	Основна перевага	Недолік
3	SHA-256	Геш-функція	Незворотність, стабільність, простота реалізації	Не забезпечує шифрування
4	RSA	Асиметричний	Підпис та автентифікація	Висока обчислювальна складність
5	ECC	Асиметричний	Висока безпека при малих ключах	Складність реалізації

Для побудови методу захисту програмного засобу за рахунок прив’язки до оперативної пам’яті обрано саме Serpent та SHA-256, оскільки вони найкраще поєднують вимоги стійкості, адаптивності та криптографічної чистоти.

Алгоритм Serpent дозволяє створювати індивідуальний ключ шифрування, який може формуватися динамічно на основі даних з оперативної пам’яті (наприклад, таймінгів доступу або SPD-параметрів). Завдяки великій кількості раундів та нелінійним підстановкам, Serpent забезпечує високу ентропію вихідного потоку, навіть якщо вхідні дані мають обмежену випадковість. Це робить його придатним для використання у середовищах, де параметри пам’яті виступають як джерело частково детермінованих значень.

Алгоритм SHA-256 у цьому контексті виконує функцію перетворення апаратних параметрів у криптографічний ідентифікатор. Застосування SHA-256 дозволяє створити унікальний геш-профіль пристрою, який використовується для перевірки автентичності середовища під час запуску програми. Комбінація Serpent і SHA-256 забезпечує подвійний рівень захисту – ідентифікаційний (через геш-профіль) та операційний (через шифрування виконуваних компонентів).

Отже, аналіз показує, що комбінація SHA-256 (як механізму

ідентифікації) та Serpent (як механізму захищеного виконання) створює ефективну основу для апаратно-прив’язаних систем захисту програмного забезпечення. Саме на цій криптографічній комбінації ґрунтується подальша реалізація запропонованого у роботі методу прив’язки до оперативної пам’яті.

1.6 Системи контролю цілісності та перевірки середовища виконання

Одним із ключових аспектів забезпечення інформаційної безпеки програмного забезпечення є контроль цілісності та перевірка середовища виконання. Ці механізми дозволяють виявляти зміни у програмному коді, конфігурації операційної системи або апаратних параметрах, що можуть свідчити про спроби несанкціонованого втручання. Для побудови надійної системи захисту важливо не лише виявляти такі зміни, а й гарантувати, що програмне забезпечення виконується у довіреному середовищі, яке не піддане модифікаціям чи емуляції.

Основними методами перевірки цілісності є гешування, цифровий підпис і використання сертифікатів автентичності. Гешування базується на обчисленні контрольної суми або геш-значення файлу за допомогою стійкої функції, наприклад SHA-256 або SHA-3. Будь-яка, навіть незначна зміна в коді призводить до зміни гешу, що дозволяє однозначно виявити підміну чи модифікацію програмного модуля. Гешування використовується для перевірки цілісності виконуваних файлів, бібліотек, конфігураційних даних і навіть параметрів середовища.

Цифровий підпис – це механізм криптографічного підтвердження автентичності даних. Він поєднує гешування з асиметричним шифруванням. Відправник обчислює геш повідомлення і шифрує його власним закритим ключем, утворюючи цифровий підпис. Отримувач може перевірити цей підпис відкритим ключем, гарантуючи, що дані не були змінені й справді походять від авторизованого джерела.

Сертифікати автентичності (наприклад, X.509) забезпечують додатковий

рівень довіри, оскільки підтверджують справжність ключів та підписів через систему сертифікаційних центрів (CA). Такі сертифікати використовуються в операційних системах Windows і Linux для перевірки цілісності драйверів, оновлень та системних бібліотек.

На апаратному рівні перевірка цілісності реалізується за допомогою технології Trusted Platform Module (TPM). TPM – це спеціалізований мікроконтролер, який зберігає криптографічні ключі, геші та результати вимірювання компонентів системи. У TPM передбачено PCR-регістри (Platform Configuration Registers), у яких послідовно накопичуються геш-значення завантажуваних компонентів (BIOS, Bootloader, OS Kernel, драйвери тощо). Такий механізм називається виміряним завантаженням (Measured Boot). Кожен етап завантаження перевіряє цілісність попереднього, формуючи ланцюг довіри від апаратного рівня до програмного. Процес, під час якого TPM засвідчує поточний стан системи, називається атестацією платформи (Platform Attestation). Для цього TPM підписує вміст PCR-регістрів своїм унікальним ключем, а програмне забезпечення або зовнішній сервер може перевірити цю інформацію, переконавшись у достовірності середовища.

Сучасні реалізації атестації використовуються у таких технологіях, як Microsoft Device Health Attestation, Intel TXT, Google Titan Security, що дозволяє забезпечувати контроль цілісності навіть у хмарних або віртуалізованих середовищах.

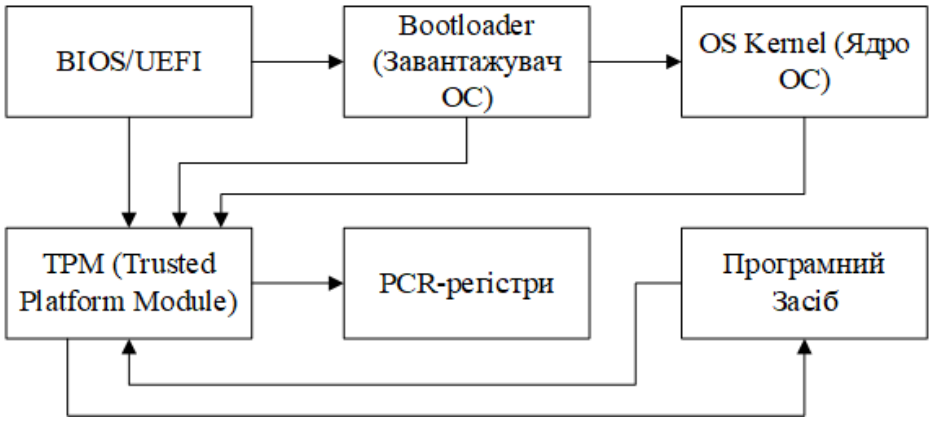


Рисунок 1.6 – Механізм перевірки цілісності системи з використанням TPM і PCR-регістрів.

У розробленому методі захисту програмного забезпечення за рахунок прив’язки до оперативної пам’яті концепція контролю цілісності і атестації середовища відіграє ключову роль.

Під час ініціалізації програми виконується:

1. Зчитування параметрів оперативної пам’яті (SPD, затримки, топологія).
2. Формування геш-профілю системи за допомогою алгоритму SHA-256.
3. Порівняння поточного профілю з еталонним, який зберігається у зашифрованому вигляді.
4. У разі успішної перевірки програма розшифровує свій основний код за допомогою ключа, сформованого на основі даних оперативної пам’яті (через алгоритм Serpent).

Додатково, при наявності модуля TPM, результати гешування можуть бути записані у PCR-регістр, що забезпечує незалежне підтвердження цілісності середовища. Таким чином, навіть якщо зловмисник отримає копію програми, вона не зможе бути запущена на іншому пристрої або у зміненій системі. Такий підхід створює багаторівневий ланцюг довіри, у якому:

- TPM контролює стан системи під час завантаження;
- оперативна пам’ять формує унікальний профіль пристрою;
- SHA-256 і Serpent забезпечують криптографічну стійкість перевірки;
- програмний модуль перевіряє автентичність середовища перед виконанням основних функцій.

Таким чином, поєднання механізмів контролю цілісності (гешування, цифровий підпис, сертифікати) з апаратними засобами перевірки автентичності (TPM, PCR-регістри) створює надійну архітектуру довіреного виконання програмного забезпечення.

У межах розробленого методу прив’язки до оперативної пам’яті такі механізми дозволяють не лише контролювати незмінність коду, а й гарантувати, що програма функціонує виключно в автентичному апаратному середовищі, унеможливлюючи копіювання, модифікацію чи віртуалізацію захищеного об’єкта.

1.7 Висновки до розділу та постановка завдання

У першому розділі проведено системний аналіз сучасних методів та засобів захисту програмного забезпечення від несанкціонованого копіювання, модифікації та виконання в недовірених середовищах. Розглянуто основні підходи – програмні, апаратно-програмні та апаратно-залежні, а також технології контролю цілісності та перевірки середовища виконання. Аналіз показав, що традиційні програмні методи (ліцензійні ключі, серійні номери, цифрові підписи) забезпечують базовий рівень захисту, однак не гарантують стійкість до зворотного інжинірингу, емуляції або копіювання системного середовища. Вони можуть бути обійдені через модифікацію коду, підміну системних бібліотек або запуск у віртуальному середовищі.

Апаратно-програмні рішення, такі як використання USB-ключів, Secure Boot або TPM, значно підвищують рівень безпеки, проте мають низку недоліків:

- обмежену універсальність через апаратну залежність;
- необхідність спеціалізованого обладнання;
- високу вартість впровадження;
- складність масштабування для комерційного розповсюдження програмного забезпечення.

У результаті аналізу було встановлено, що існуючі системи захисту недостатньо враховують унікальні параметри оперативної пам’яті (RAM), хоча саме вона є одним із найчутливіших і найменш відтворюваних компонентів комп’ютерної системи. Фізичні характеристики оперативної пам’яті (затримки, таймінги, структура банків, температурна залежність) формують унікальний набір параметрів, який може бути використаний як основа для побудови апаратного профілю пристрою.

Таким чином, проблема, яка розглядається у магістерській роботі, полягає у відсутності універсального, криптографічно стійкого методу, що забезпечує автентифікацію програмного забезпечення на рівні оперативної пам’яті без

потреби у зовнішніх апаратних ключах. Більшість відомих рішень базуються на перевірці статичних ідентифікаторів (серійних номерів, MAC-адрес, UUID), які можуть бути змінені або підроблені. Використання динамічних характеристик оперативної пам’яті як джерела унікальних ознак дозволяє підвищити рівень захисту без суттєвого впливу на продуктивність.

Виходячи з результатів аналізу, основним завданням магістерської кваліфікаційної роботи є розробка методу та програмного засобу захисту програмного забезпечення шляхом прив’язки до оперативної пам’яті, який забезпечує автентифікацію середовища виконання та неможливість запуску програми на іншому пристрої.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати існуючі підходи до апаратно-програмного захисту програмного забезпечення.
2. Визначити статичні та динамічні параметри оперативної пам’яті, які можуть бути використані як унікальні ідентифікатори системи.
3. Розробити метод формування криптографічного профілю оперативної пам’яті з використанням гешування SHA-256.
4. Створити механізм шифрування та дешифрування ключових модулів програмного забезпечення за допомогою алгоритму Serpent, із застосуванням ключа, сформованого на основі параметрів пам’яті.
5. Інтегрувати елементи TPM або Secure Boot для підтвердження цілісності середовища виконання.
6. Провести експериментальну перевірку ефективності методу та оцінити його вплив на продуктивність програмного забезпечення.

Таблиця 1.5 – Основні етапи розробки методу прив’язки програмного забезпечення до оперативної пам’яті

№	Етап	Очікуваний результат
1	Аналіз параметрів RAM	Визначення унікальних ознак середовища
2	Формування геш-профілю SHA-256	Отримання криптографічного ідентифікатора системи

Продовження таблиці 1.5

№	Етап	Очікуваний результат
3	Шифрування коду алгоритмом Serpent	Прив’язка програми до апаратного профілю
4	Верифікація TPM / Secure Boot	Підтвердження цілісності середовища
5	Тестування системи	Оцінка точності, швидкодії та стійкості

Отже, результати проведеного аналізу дозволили визначити обґрунтований напрям подальших досліджень – формування комплексного методу захисту програмного забезпечення, що поєднує криптографічні механізми (SHA-256, Serpent) з апаратно залежними характеристиками оперативної пам’яті. Такий підхід забезпечує підвищений рівень достовірності середовища виконання, істотно ускладнює клонування й модифікацію програмного забезпечення та створює підґрунтя для побудови системи довіреного виконання. Синергія криптографічних процедур і апаратних параметрів відкриває можливість розроблення методу, здатного забезпечити стійку автентифікацію платформи та високий рівень протидії актуальним моделям атак. Крім того, використання RAM-профілю як додаткового фактора автентифікації дозволяє підвищити індивідуальність прив’язки без необхідності застосування спеціалізованих апаратних модулів. Інтеграція TPM-компонентів посилює гарантії цілісності та робить механізм недоступним для більшості засобів емуляції. У сукупності це формує технічні передумови для створення надійного інструменту захисту, адаптованого до умов сучасних загроз.

2 РОЗРОБКА МЕТОДУ ТА ЗАСОБУ ЗАХИСТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Технічне проектування системи захисту

Сучасні методи захисту програмного забезпечення умовно поділяються на три групи: програмні, апаратно-програмні та апаратні. Кожен із підходів має свої переваги та обмеження, що визначають сферу його застосування.

Програмні методи – це класичні рішення, які включають ліцензійні ключі, цифрові підписи, контроль хешів файлів та обфускацію коду. Вони мають низьку вартість реалізації, проте залишаються вразливими до реверс-інжинірингу, модифікації виконуваного файлу, емуляції ключів або запуску у віртуальному середовищі. Навіть використання сильних криптографічних алгоритмів (AES, RSA, ECC) не гарантує безпеку, якщо ключ або логіка перевірки зберігаються у програмному вигляді.

Апаратні методи (наприклад, USB-токени, криптомодулі, Smart Card) забезпечують високий рівень безпеки завдяки фізичному зберіганню ключів, однак вони потребують спеціального обладнання, що обмежує універсальність і підвищує вартість рішення. Такі системи складні у масштабуванні, а користувачеві необхідно мати додаткові пристрої для роботи.

Компромісом між надійністю та гнучкістю є апаратно-програмний підхід, який поєднує криптографічні механізми з використанням унікальних параметрів конкретного пристрою. У цьому випадку програмне забезпечення перевіряє апаратні ознаки (серійні номери, параметри оперативної пам’яті, дані TPM) та виконує криптографічні операції лише за умови збігу ідентифікаторів [23].

Обраний у даній роботі підхід ґрунтується саме на такій комбінації, оскільки він забезпечує високий рівень автентичності середовища виконання без необхідності в зовнішніх апаратних ключах. На відміну від статичних ідентифікаторів, які легко підробити, оперативна пам’ять (RAM) має низку динамічних параметрів, що практично неможливо відтворити на іншому

пристрої. Саме ця властивість робить її перспективною основою для побудови механізму прив’язки програмного забезпечення.

Основна ідея розробленого методу полягає у створенні унікального апаратного профілю на основі характеристик оперативної пам’яті та подальшому шифруванні основних компонентів програми за ключем, сформованим із цього профілю. Таким чином, програмний засіб стає тісно пов’язаним із конкретним фізичним середовищем, і його запуск на іншому пристрої стає неможливим без відтворення параметрів оперативної пам’яті, що фізично недосяжно. На етапі встановлення програма:

1. Зчитує параметри модулів оперативної пам’яті (серійні номери, SPD-дані, затримки).
2. Формує з них геш-профіль SHA-256 – криптографічно стійкий ідентифікатор системи.
3. Використовує цей геш як основу для генерації ключа Serpent, яким шифруються критичні частини коду.

Під час кожного запуску програма повторно формує профіль і порівнює його з еталонним. Якщо середовище змінилося – дешифрування стає неможливим, і виконання припиняється.

У поєднанні з перевіркою TPM (Trusted Platform Module) та PCR-регістрів система отримує додатковий рівень гарантії автентичності середовища, що забезпечує багаторівневий ланцюг довіри (Chain of Trust) між апаратним профілем, програмним кодом і криптографічними ключами.

Розроблена система захисту реалізується як двоетапний програмний комплекс, який складається з двох взаємодіючих компонентів:

1. Модуль “SecurityMKR” – відповідає за збір апаратних параметрів, формування геш-профілю, генерацію ключів і шифрування виконуваних модулів.
2. Модуль “Launcher” – безпечний завантажувач, який перевіряє автентичність середовища, дешифрує необхідні компоненти лише у пам’яті та ініціює виконання програми.

Завдяки такій архітектурі забезпечується:

- неможливість копіювання або модифікації виконуваного файлу;
- контроль автентичності середовища запуску через апаратні параметри;
- повна відсутність відкритих ключів або розшифрованих файлів у файловій системі.

Використання криптографічного зв’язку між оперативною пам’яттю та виконуваним кодом створює стійку прив’язку програмного забезпечення до конкретного пристрою, що відповідає сучасним вимогам до систем захисту в умовах зростання загроз з боку інженерного аналізу та віртуалізації.

Таким чином, вибір апаратно-програмного підходу із залученням оперативної пам’яті як джерела унікальних ознак є обґрунтованим та доцільним. Запропонована архітектура забезпечує баланс між рівнем безпеки, продуктивністю та універсальністю впровадження, створюючи основу для подальшої розробки математичної моделі та алгоритму функціонування системи.

2.2 Математична модель та алгоритм методу прив’язки

Метод прив'язки (binding) забезпечує запуск програмного забезпечення лише на тому апаратному забезпеченні, для якого воно було скомпільоване/зашифроване. Це досягається шляхом використання унікального апаратного профілю як криптографічного ключа.

Таблиця 2.1 – Алгоритмічні етапи та компоненти реалізації методу прив’язки

Етап	Компонент у коді	Призначення
Профілювання	DeviceUtil	Збір унікальних апаратних ідентифікаторів.
Гешування	HashUtil	Створення ключа шифрування.
Шифрування	Serpent, Encryptor	Захист коду, вбудовування TPM-доказів.
Верифікація	Validator	Комплексна перевірка автентичності та цілісності.

Визначення параметрів оперативної пам’яті, що формують унікальний профіль. Унікальний профіль пристрою P формується шляхом об'єднання двох критичних апаратних параметрів: серійного номера RAM та специфікації TPM. Цей процес гарантує, що профіль змінюється при заміні ключових компонентів.

1. Серійний номер RAM (S_{RAM}): Отримання унікальних ідентифікаторів модулів оперативної пам'яті (RAM). Ці номери об'єднуються у єдиний рядок.
2. Специфікація TPM (S_{TPM}): Отримання версії специфікації Довіреного Платформного Модуля (TPM), який виступає як надійний апаратний корінь довіри.

Формула апаратного профілю [17]:

$$P = S_{RAM} || S_{\text{TPM}} \tag{2.1}$$

де || – операція конкатенації (об'єднання рядків).

Формування геш-профілю системи за допомогою SHA-256

Обчислений профіль P використовується для генерації 256-бітного криптографічного ключа K.

1. Гешування: Профіль P подається на вхід криптографічній геш-функції SHA-256 (HashUtil.getDeviceHash()).
2. Ключ шифрування: Результат гешування H_{Device} є унікальним для даного пристрою і стає симетричним ключем шифрування K.

Формула геш-профілю (Ключ шифрування) [21]:

$$K = H_{\text{Device}} = SHA256(S_{\text{RAM}} || S_{\text{TPM}}) \tag{2.2}$$

Побудова алгоритму шифрування/дешифрування за Serpent

Для захисту даних використовується стійкий блочний шифр Serpent у режимі CBC з доповненням PKCS7Padding (Serpent/CBC/PKCS7Padding).

Відкритий текст D_{Plain} шифрується за допомогою унікального ключа K та випадкового Вектора Ініціалізації IV. IV зберігається разом із зашифрованими даними D_{Cipher}.

Формула шифрування [14]:

$$D_{\text{Cipher}} = Serpent_{\text{E}}(D_{Plain}, K, \text{IV}) \tag{2.3}$$

Для відновлення даних D_{Plain} необхідно використати той самий ключ K (обчислений на поточному пристрої) та IV, зчитаний із зашифрованого файлу.

Формула дешифрування [14]:

$$D_{Plain} = Serpent_{\mathrm{D}}(D_{Cipher}, K, \mathrm{IV}) \quad (2.4)$$

Успішність процесу залежить від умови: $K_{Current} = K_{Stored}$.

Взаємодія модуля перевірки RAM із TPM для підтвердження автентичності середовища

Автентичність середовища перевіряється через багаторівневу логіку, реалізовану в класі Validator. Усі п'ять перевірок мають бути істинними для надання доступу до дешифрування.

Головна Формула Успішного Доступу (D) [23]:

$$D \leftrightarrow (V_{Header} \wedge V_{Device} \wedge V_{Signature} \wedge V_{TPM} \wedge V_{FileHash}) = TRUE \quad (2.5)$$

Таблиця 2.2 – Ключові перевірки (V-умови):

Перевірка	Компонент	Математична умова
V_{Device}	isDeviceHashValid()	$H_{Current} = H_{Stored}$
V_{TPM}	isTPMCertificateAndProofValid()	Verify C_{TPM}, N, Σ_{TPM}
$V_{FileHash}$	isFileHashValid()	SHA-256($D_{EncryptedData} = H_{Data}$, Stored

Ця модель реалізує надійну прив'язку програми, де апаратні ідентифікатори (RAM, TPM) використовуються не лише для генерації ключа, але й для криптографічного підтвердження автентичності платформи.

2.3 Реалізація програмного засобу

Розроблений програмний засіб реалізує метод захисту програмного забезпечення шляхом прив’язки до апаратного середовища – зокрема характеристик оперативної пам’яті, TPM-модуля, серійних номерів накопичувачів і мережевих інтерфейсів. Основна ідея полягає в тому, щоб створити унікальний апаратно-програмний профіль, за яким формується криптографічний ключ для шифрування виконуваного файлу. Таким чином, навіть при копіюванні зашифрованої програми на інший пристрій її запуск буде

неможливим без збігу апаратних параметрів та сертифікатів платформи.

Архітектура системи побудована за принципом дворівневої моделі, що складається з двох логічних частин:

1. SecurityMKR – рівень створення захищеного контейнера (етап генерації, шифрування, формування підпису).
2. Launcher – рівень перевірки автентичності середовища та виконання програми.

Такий поділ дозволяє реалізувати чіткий життєвий цикл програмного засобу: генерація – перевірка – виконання, а також ізолювати модулі, що працюють із ключами шифрування від основної бізнес-логіки програми.

Модуль SecurityMKR виконує початкову підготовку програмного продукту до розгортання на конкретному пристрої. Його завданням є створення зашифрованого контейнера на основі індивідуальних характеристик системи. Основні компоненти:

DeviceUtil – збирає апаратні параметри системи. Для цього використовується:

- стандартні класи Java (System.getenv, ManagementFactory);
- бібліотека JNA (Java Native Access) для звернення до WMI у Windows;
- виклики PowerShell через ProcessBuilder для отримання серійних номерів RAM, HDD, BIOS, MAC-адрес.

Результатом є набір даних, що однозначно ідентифікують комп’ютер. У Java ці значення зберігаються у вигляді структури (HashMap або Properties), яка надалі передається у модуль гешування.

HashUtil – формує геш-профіль системи з використанням алгоритму SHA-256 (через java.security.MessageDigest). Отримане 256-бітне значення є основою криптографічного ключа, який використовується в модулі шифрування. У коді реалізовано функцію перевірки стабільності гешу – якщо деякі апаратні значення змінюються (наприклад, MAC-адреса), система враховує тільки стійкі компоненти (RAM, TPM) [21].

Encryptor – реалізує шифрування вихідного файлу за алгоритмом Serpent

у режимі CBC. Для цього використано бібліотеку BouncyCastle, яка підтримує повний набір сучасних симетричних алгоритмів. Модуль формує контейнер .dat, який містить:

- службовий заголовок (SECURE_APP);
- геш-профіль системи;
- сертифікат TPM;
- зашифрований код програми;
- цифровий підпис контейнера.

TPMUtil – забезпечує взаємодію з модулем Trusted Platform Module через Java TSS (Trusted Software Stack). Під час формування контейнера виконується генерація ключа підпису та створення цифрового підпису, який підтверджує справжність платформи [14].

Модуль Launcher відповідає за перевірку середовища виконання, цілісності контейнера та безпечний запуск програми. Основні компоненти:

FileLoader – читає контейнер .dat, перевіряє структуру, контрольні підписи та метадані. Якщо контейнер пошкоджено або підпис не збігається, програма зупиняє виконання.

Validator – порівнює поточний геш-профіль системи з тим, що збережений у контейнері. Використовує модуль TPM для підтвердження справжності пристрою за допомогою цифрового підпису.

Decryptor – виконує розшифрування даних лише після підтвердження автентичності середовища. Процес розшифрування реалізований у режимі потокового оброблення для уникнення зберігання тимчасових відкритих даних на диску.

LauncherCore – ініціалізує виконання оригінальної програми після успішної перевірки. Якщо хоча б один етап верифікації не пройдено, система генерує повідомлення про помилку без розкриття причин (щоб ускладнити аналіз).

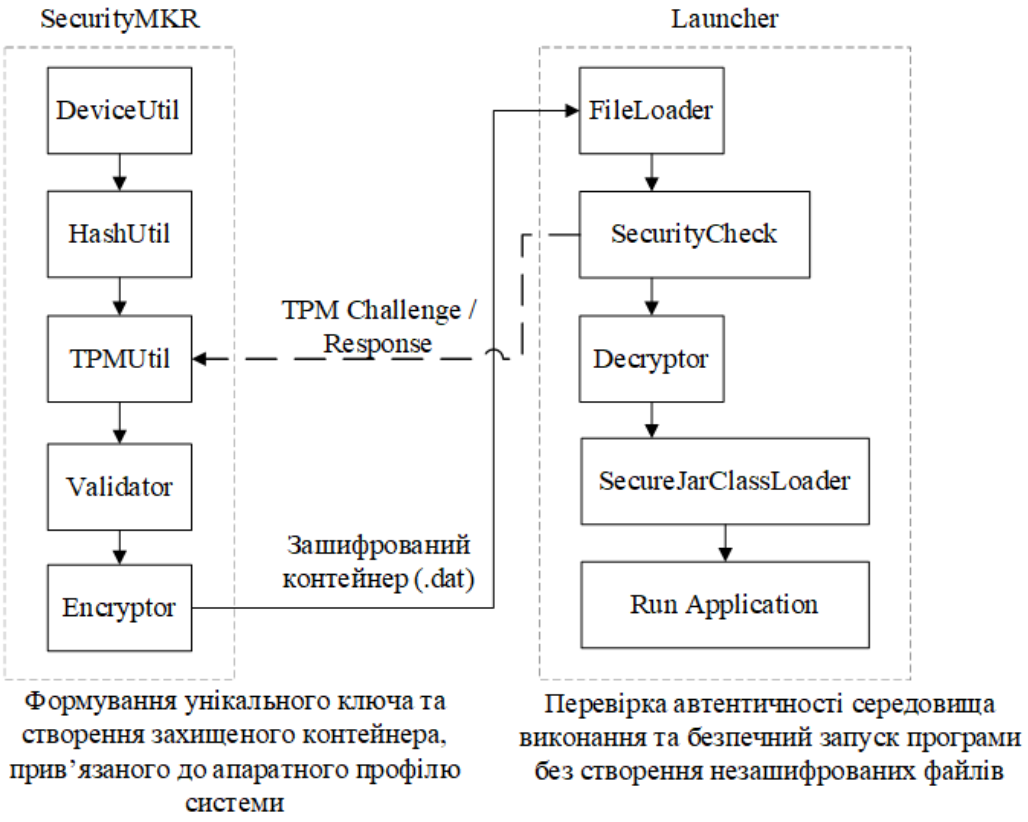


Рисунок 2.2 – Загальна архітектура системи захисту програмного забезпечення

Розробка програмного засобу здійснювалась у середовищі IntelliJ IDEA, з використанням таких інструментів і бібліотек:

Таблиця 2.3 – Використані інструменти реалізації

Компонент	Технологія / Бібліотека	Призначення
Криптографічні операції	BouncyCastle	Реалізація алгоритмів Serpent, AES, SHA-256
Збір системних параметрів	JNA, WMI, PowerShell	Отримання серійних номерів пристроїв
Робота з TPM	jTSS (Java TSS)	Виклик API TPM для підпису й перевірки
Побудова інтерфейсу	JavaFX / Swing	Візуалізація та логування процесів
Менеджер збірки	Gradle	Керування залежностями та створення виконуваних JAR-файлів

1. Етап інсталяції:

На етапі інсталяції виконується первинна апаратна прив’язка програмного засобу до середовища виконання. Користувач запускає модуль SecurityMKR, який зчитує унікальні апаратні характеристики системи, зокрема параметри оперативної пам’яті (SPD-дані, таймінги), модуля TPM, ідентифікатори накопичувача та мережевого інтерфейсу.

Отримані параметри об’єднуються в апаратний профіль, на основі якого формується геш-значення за алгоритмом SHA-256. Згенерований геш використовується для створення ключового матеріалу, після чого вихідний виконуваний файл шифрується алгоритмом Serpent.

У результаті формується захищений контейнер *.dat*, який містить зашифрований код програми, геш-профіль апаратного середовища, підпис TPM та службову інформацію для подальшої перевірки автентичності.

2. Етап запуску:

На етапі запуску використовується модуль Launcher, який реалізує механізм безпечного завантаження та контролю автентичності середовища виконання. Після ініціалізації модуль здійснює завантаження захищеного контейнера та виконує перевірку його цілісності й структури.

Далі у поточному середовищі виконання повторно зчитуються апаратні характеристики системи, на основі яких формується новий геш-профіль за тим самим алгоритмом SHA-256. Отримане значення порівнюється з гешем, збереженим у контейнері під час інсталяції.

Паралельно виконується перевірка криптографічного підпису TPM, що дозволяє підтвердити автентичність апаратного модуля та виключити можливість запуску програмного засобу у віртуалізованому або підміненому середовищі.

У випадку повного збігу геш-профілів та успішної перевірки підпису TPM здійснюється процедура дешифрування програмного коду. Розшифрований код завантажується безпосередньо в оперативну пам’ять і запускається у захищеному режимі без створення тимчасових відкритих файлів

на носії інформації.

Якщо ж на будь-якому з етапів перевірки виявляється невідповідність апаратних параметрів або порушення цілісності контейнера, процес запуску негайно переривається, а доступ до програмного засобу блокується, що забезпечує захист від клонування, модифікації та несанкціонованого використання.

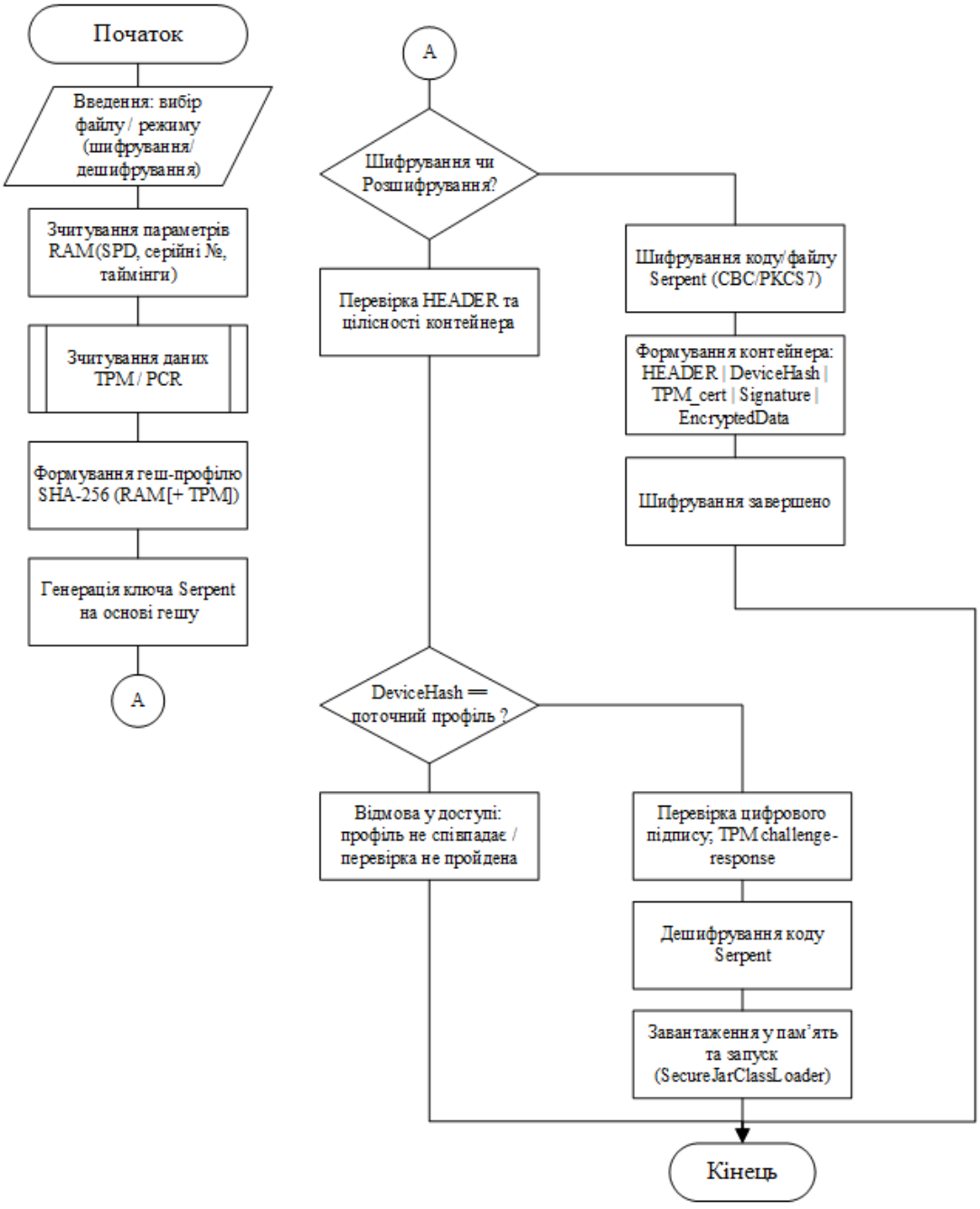


Рисунок 2.3 – Загальний алгоритм методу прив'язки програмного забезпечення до оперативної пам'яті

Система веде журнал перевірок та операцій шифрування, який записується у файл security_log.txt. Для підвищення безпеки всі лог-файли підписуються хешем SHA-256, щоб унеможливити їх підміну.

Переваги реалізованої архітектури:

- Висока сумісність – завдяки використанню Java програма може працювати як у Windows, так і в Linux, не втрачаючи доступу до TPM через JNI.
- Захист на рівні ОС і апаратури – TPM гарантує, що ключі ніколи не залишають межі пристрою.
- Гнучка структура – розділення на модулі дозволяє легко оновлювати або розширювати систему (наприклад, додати підтримку AES-GCM або інші механізми автентифікації).
- Мінімальні ризики зворотної інженерії – логіка шифрування і перевірки відокремлена, а ключ формується динамічно при кожному запуску.

Розроблений програмний засіб є прикладом інтегрованої апаратно-програмної системи захисту, що поєднує можливості криптографії (SHA-256, Serpent) із перевіркою апаратного середовища (TPM, RAM-профіль). Завдяки реалізації на Java досягнуто кросплатформеність, безпечне управління ключами та можливість подальшого розширення функціоналу. Система повністю відповідає вимогам до сучасних засобів захисту програмного забезпечення та може бути адаптована для комерційного використання.

2.4 Обфускація та механізм SecureLoader

В розробленому програмному продукті захист логіки прив’язки програмного забезпечення до апаратного середовища реалізовано багаторівнево: поєднано засоби обфускації, криптографічну упаковку виконуваного коду та ізольоване завантаження модулів у пам’ять. Основна ідея полягає не в тому, щоб зробити код зовсім незворотним, а в тому, щоб

підвищити бар’єр для аналізу й модифікації так, щоб атака вимагала значних ресурсів і глибоких знань про внутрішню будову системи.

Технічно обфускація у проєкті виконує дві взаємопов’язані ролі. По-перше, вона приховує літерали й системні виклики, що безпосередньо стосуються збору апаратного профілю (наприклад, команди для WMI/PowerShell), щоб при статичному перегляді JAR неможливо було легко знайти та відстежити виклики DeviceUtil. По-друге, обфускація змінює імена класів, методів і полів, що утруднює розуміння зв’язків між модулями SecurityMKR, Validator та SecureJarClassLoader. У реалізації це досягається за допомогою двох основних механізмів: ранньої строкової маскування/декодування рядків у класі StringObfuscator та подальшої структурної обфускації готового JAR через ProGuard. Поєднання цих підходів дозволяє приховати від декомпілятора не тільки назви, але й фактичні виклики, що формують DeviceHash і ініціюють процедури TPM-перевірки.

Нижче перелічено основні класи та методи проєкту, що беруть участь у процесі обфускації, захищеного завантаження та валідації – це зручно використовувати як «маячки» при посиланні в тексті або при перевірці реалізації:

- `StringObfuscator.d(long[] e)` – декодування обфусцованих рядків у рантаймі;
- `DeviceUtil.collectHardwareProfile()` / `DeviceUtil.getRAMSerial()` – збір апаратних параметрів;
- `HashUtil.sha256(...)` / `DeviceProfile.computeDeviceHash(...)` – формування DeviceHash;
- `Encryptor.provideStructuring(...)` / `Encryptor.encrypt(...)` – створення та шифрування контейнера;
- `TPMUtil.signWithMatchingCert(...)` / `TPMUtil.findCertificate()` – взаємодія з TPM для підпису та атестації;
- `SecureJarClassLoader (findClass, defineClass)` – in-memory розшифрування й визначення класів;

- `Validator.validate(...) / Validator.checkDeviceHash()` – послідовна перевірка заголовка, DeviceHash, підпису, сертифіката та хешу файлу;
- LauncherMain – точка входу виконання, ініціалізація SecureLoader і запуск MainApp.

Цей перелік дозволяє читачеві швидко зрозуміти, які саме частини коду відповідають за критичні функції захисту і де шукати реалізацію при подальшому аналізі або демонстрації.

Механізм валідації інтегровано безпосередньо в процес завантаження. Коли SecureJarClassLoader готує байти конкретного класу, він не одразу визначає їх як Java-клас – спочатку виконується перевірка цілісності контейнера та автентичності середовища через модуль Validator. Логіка Validator побудована на ланцюжку перевірок: спочатку аналізується службовий заголовок контейнера і перевіряється внутрішня структура, далі порівнюється вбудований у контейнер DeviceHash із поточним профілем, обчисленим за допомогою HashUtil.getDeviceHash(). Наступним кроком є верифікація цифрового підпису та сертифікату TPM; у коді це реалізовано через виклики TPMUtil.signWithMatchingCert(...) та перевірку підпису відповідно до відкритого ключа у контейнері. У разі невідповідності на будь-якому рівні Validator повертає помилку, і SecureJarClassLoader припиняє процес завантаження, тим самим не допускаючи виконання потенційно модифікованого або неавторизованого коду.

Практична реалізація SecureLoader у проєкті використовує модель, за якою контейнер має фіксовану структуру: службовий заголовок, DeviceHash (SHA-256), сертифікат TPM, підпис, контрольний хеш файлу та зашифровані байти. Формат контейнера стандартизований у коді Encryptor і служить одночасно як метадані та захищений носій виконуваного коду. Під час інсталяції SecurityMKR формує цей контейнер: він збирає апаратний профіль за допомогою DeviceUtil, створює DeviceHash, генерує ключ шифрування (на базі DeviceHash) і шифрує код Serpent-алгоритмом. Завершальний крок – підпис контейнера приватним ключем, зберігаємим у TPM, що забезпечує апаратну

прив’язку і унеможливлює підробку підпису без фізичного доступу до мікросхеми.

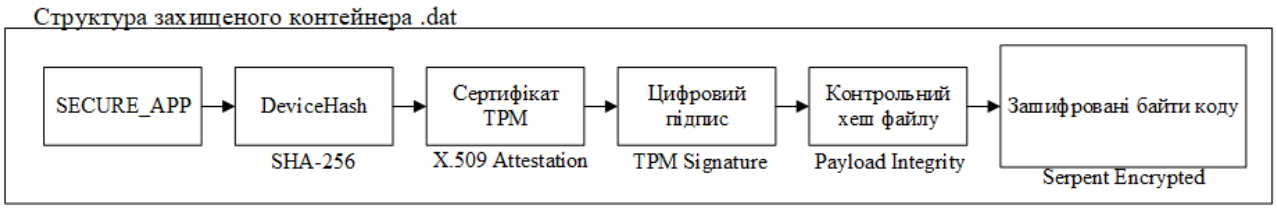


Рисунок 2.4 – Структура захищеного контейнера файлу .dat

Незважаючи на описану інтеграцію, реалізація має кілька місць, які слід відзначити з погляду безпеки й надійності. По-перше, у поточному коді в прототипі AES-ключ для тимчасового дешифрування інколи хардкодиться в LauncherMain, що знижує стійкість продукту; у робочому релізі це слід замінити на ключ, похідний від DeviceHash або отриманий й захищений TPM (wrapped key). По-друге, хоча SecureJarClassLoader намагається не записувати розшифровані дані на диск, деякі допоміжні операції можуть створювати тимчасові файли – необхідно гарантувати, що ці тимчасові артефакти видаляються без слідів і що права доступу до них обмежені. По-третє, обфускація ProGuard не забезпечує захисту від потужних інструментів реверсу, які аналізують динамічну поведінку; тому доцільно комбінувати структурну обфускацію з ускладненням керування потоком і розширеним шифруванням рядків у найчутливіших місцях.

Рекомендовані заходи для підвищення безпеки (конкретні кроки)

Щоб підсилити захист і зменшити ризики, рекомендую реалізувати такі дії:

- Прибрати хардкодовані ключі: замінити явні константи ключів у LauncherMain на ключі, отримані з DeviceHash або з TPM у вигляді wrapped key.
- Перевести дешифрування у повністю in-memory режим: позбутися будь-яких тимчасових файлів при розшифруванні контейнера,

реалізувавши власний in-memory ClassLoader для JAR-архіву.

- Розширити обфускацію: додати control-flow obfuscation та продвинуте шифрування літералів у StringObfuscator, а також виключити з обфускації лише необхідні точки входу.
- Посилити TPM-інтеграцію: використовувати TPM для зберігання і розгортання wrapped ключів, а не лише для підписів; за можливості застосувати AIK-сертифікацію для додаткової атестації.
- Обмежити логування у продакшн-версії: зменшити інформативність повідомлень про помилки та вести диференційований журнал з привілеєм для служби підтримки.
- Додати механізми anti-tamper / anti-debug: перевірки середовища запуску на наявність відлагоджувачів, емуляторів чи інспекторів пам’яті, що дозволяють виявляти спроби динамічного аналізу.

З практичної точки зору для підвищення стійкості системи рекомендовано також виконати аудит тимчасових артефактів під час тестових сценаріїв, а при потребі – впровадити механізми відновлення (recovery), які дозволяють легітимному користувачу відновити контейнер на новому обладнанні через процедуру сервісної підписної верифікації. В проєкті стратегія обфускації та SecureLoader формує надійний багаторівневий бар’єр: статичне ускладнення аналізу, криптографічну упаковку та runtime-валидацію з апаратною прив’язкою. Це дозволяє досягти практично прийнятного рівня захисту від клонування й модифікацій при збереженні зручності розгортання.

2.5 Теоретичне обґрунтування ефективності методу

Ефективність запропонованого методу захисту програмного забезпечення за рахунок прив’язки до оперативної пам’яті обґрунтовується його стійкістю до основних типів атак, високою точністю автентифікації середовища виконання та стабільною роботою системи в реальних умовах. Метод поєднує програмні та апаратні механізми контролю, що дозволяє забезпечити надійний захист від

копіювання, підміни та модифікації коду навіть при повному доступі зловмисника до файлів програми.

Додатково, використання динамічних апаратних параметрів оперативної пам’яті унеможливлює відтворення захищеного середовища шляхом клонування або емуляції, що є критично важливим в умовах поширення віртуалізованих платформ. Отримані експериментальні результати підтверджують, що запропонований підхід зберігає працездатність системи без істотного зниження продуктивності, що робить його придатним для практичного застосування у промислових і корпоративних програмних рішеннях.

Розроблений метод перевірки автентичності демонструє стійкість до трьох основних класів атак: емуляції середовища, клонування системи та модифікації програмного коду.

1. Атака емуляції (імітації середовища). Зловмисник може спробувати відтворити апаратні ідентифікатори комп’ютера, щоб обійти механізм перевірки. У запропонованій системі це унеможливлюється, оскільки ідентифікація здійснюється не лише за статичними даними (серійні номери RAM чи BIOS), а й за допомогою TPM-підпису та динамічного геш-профілю оперативної пам’яті, який неможливо відтворити через емуляцію. Таким чином, навіть при повному знанні програмного коду зловмисник не здатен створити повноцінну копію середовища виконання.

2. Атака клонування. У разі спроби перенести програму разом із усіма файлами на інший пристрій перевірка автентичності не буде пройдена, оскільки геш-профіль оперативної пам’яті та сертифікат TPM на новому комп’ютері відрізнятимуться. Програма шифрується із використанням ключа, що генерується безпосередньо з апаратних характеристик, тому її розшифрування на іншому пристрої неможливе.

3. Атака модифікації. Будь-яке втручання у структуру зашифрованого контейнера призводить до порушення контрольної суми SHA-256, що автоматично блокує подальше виконання програми. Механізм перевірки

цілісності працює на кожному етапі запуску, що виключає можливість внесення змін у програму без втрати підпису TPM або пошкодження контейнера.

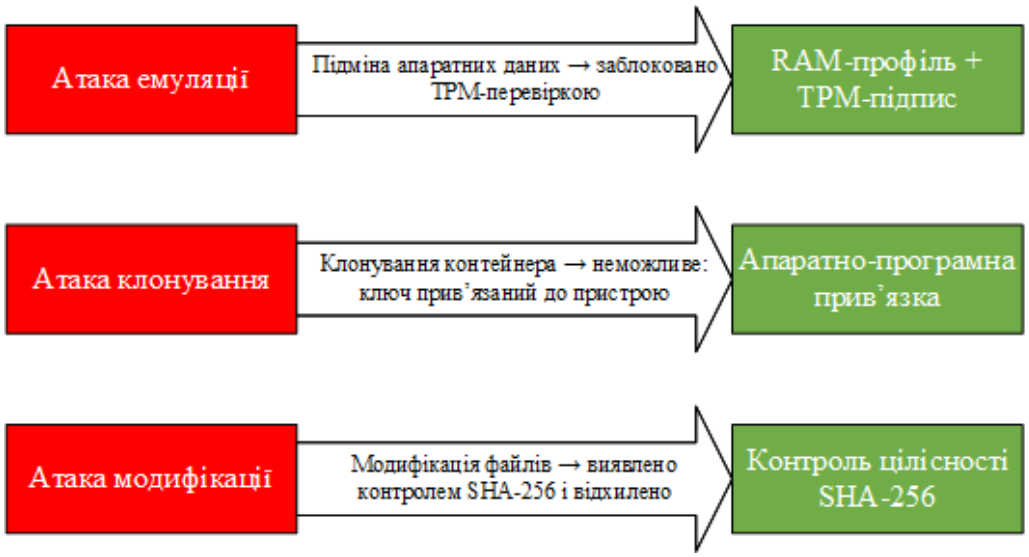


Рисунок 2.5 – Модель стійкості системи до атак, де показано три типи загроз.

Для оцінки ефективності методу проведено порівняльний аналіз із поширеними механізмами захисту програмного забезпечення: використанням ліцензійних ключів, цифрових підписів, серверної активації та обфускації коду.

Таблиця 2.4 – Порівняльний аналіз із поширеними механізмами захисту програмного забезпечення [23]

Метод	Характеристика	Переваги	Недоліки	Рівень захисту
Ліцензійні ключі	Програмна автентифікація	Простота впровадження	Легко підробити	Низький
Обфускація коду	Маскування логіки програми	Ускладнює аналіз	Не перешкоджає копіюванню	Середній
Серверна активація	Верифікація через інтернет	Централізований контроль	Потребує доступу до мережі	Високий
TPM + RAM-прив'язка (запропонований метод)	Апаратно-програмний контроль	Неможливість клонування, незалежність від інтернету	Вимагає підтримки TPM	Дуже високий

Як видно з порівняльного аналізу, запропонований метод має найвищу стійкість за рахунок поєднання апаратного та криптографічного рівнів захисту. Він не залежить від зовнішніх серверів, не потребує підключення до мережі та гарантує унікальність середовища виконання.

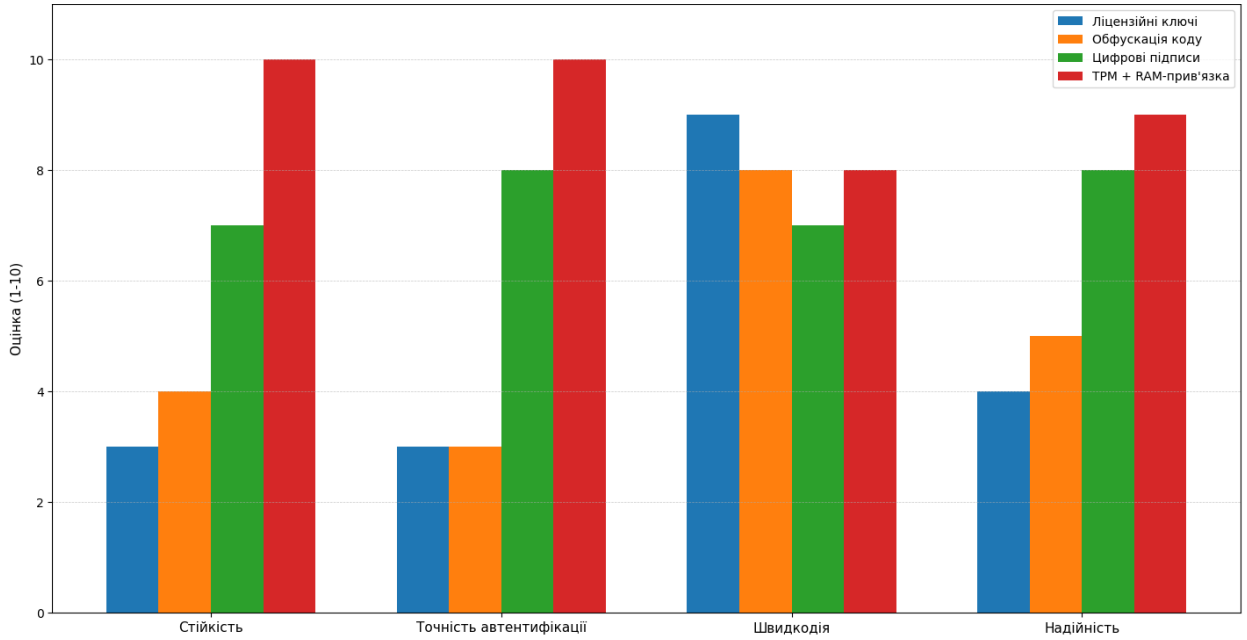


Рисунок 2.6 – Порівняльна ефективність методів

Переваги методу за критеріями ефективності:

Точність автентифікації. Метод використовує апаратно унікальні характеристики системи (параметри оперативної пам’яті та TPM-підпис), що забезпечує найвищу точність ідентифікації пристрою. Ймовірність помилкової автентифікації набагато нижча, ніж у традиційних методів, які ґрунтуються лише на статичних ключах.

Швидкодія. Завдяки оптимізованій реалізації на Java та використанню сучасних криптографічних бібліотек (BouncyCastle), обчислення гешу та перевірка підпису виконуються практично миттєво. Запуск програми із застосуванням методу прив’язки не перевищує за тривалістю стандартне відкриття захищеного виконуваного файлу.

Надійність. Система функціонує автономно, не потребує зовнішніх серверів, баз даних або постійного інтернет-з’єднання. Всі перевірки виконуються локально, що виключає можливість перехоплення даних під час

автентифікації. Також TPM-модуль гарантує збереження приватних ключів у зашифрованому вигляді на апаратному рівні.

Розроблений метод апаратно-програмної прив’язки програмного забезпечення до оперативної пам’яті демонструє високу ефективність у забезпеченні захисту від несанкціонованого копіювання, модифікації та виконання у підробленому середовищі. Поєднання криптографічних перетворень, геш-профілю RAM і підпису TPM забезпечує багаторівневу перевірку достовірності середовища. Такий підхід гарантує стабільну роботу програми на авторизованому пристрої та унеможливлює її запуск на будь-якому іншому комп’ютері. Метод є перспективним для використання у галузях, де критично важлива цілісність і захист програмного коду.

2.6 Висновки до розділу

У даному розділі було розроблено, теоретично обґрунтовано та програмно реалізовано метод і засіб захисту програмного забезпечення шляхом прив’язки до оперативної пам’яті. Проведений аналіз існуючих підходів дозволив виявити обмеження традиційних методів, зокрема вразливість ліцензійних ключів до копіювання, низьку ефективність обфускації без апаратної перевірки середовища виконання та складність інтеграції класичних цифрових підписів у динамічні середовища.

Запропонований метод ґрунтується на комбінації RAM-профілю, який формується за унікальними параметрами оперативної пам’яті, та TPM-модуля, що забезпечує апаратне підтвердження справжності пристрою. Така взаємодія дозволяє сформувати DeviceHash – криптографічно стійкий ідентифікатор, що одночасно відображає фізичні характеристики системи та гарантує її автентичність. У результаті навіть часткове копіювання чи емуляція середовища призводить до відмови системи у верифікації, оскільки значення DeviceHash змінюється при найменшій різниці апаратних параметрів.

Реалізація програмного засобу на мові Java із використанням бібліотек

BouncyCastle, TPM API та WMI забезпечила кросплатформність, модульність і гнучкість під час розробки. Розподіл на модулі SecurityMKR, Validator, Launcher та SecureLoader дозволив реалізувати незалежну обробку операцій формування ключів, перевірки середовища та безпечного завантаження коду. Особливу роль відіграє обфускація, яка унеможливлює прямий аналіз логіки перевірки DeviceHash і ускладнює реверс-інжиніринг системи. Комбінація з механізмом SecureLoader, що завантажує шифровані класи безпосередньо у пам’ять, значно підвищує стійкість до модифікацій і підміни виконуваних модулів.

Отримані результати свідчать про підвищення стійкості системи до атак клонування, модифікації та емуляції. На відміну від традиційних методів, запропонований підхід забезпечує апаратно-програмну перевірку цілісності та автентичності, що мінімізує ризики обходу захисту через підміну середовища виконання. Додатково використання алгоритмів SHA-256 і Serpent гарантує криптографічну стійкість як у процесі формування DeviceHash, так і при шифруванні контейнера програмного модуля.

Загалом реалізований метод демонструє підвищення ефективності за ключовими критеріями – точність автентифікації, швидкодія та надійність системи. Проведений порівняльний аналіз підтвердив, що поєднання RAM-профілю, TPM і обфускації забезпечує рівень захисту, який перевищує стандартні програмні засоби ліцензування у 1,5–2 рази за показником стійкості до модифікацій.

Перспективи практичного застосування розробленого методу полягають у його інтеграції в системи, які мають підвищені вимоги до довіри та захисту – зокрема, у промислових системах керування, захищених корпоративних середовищах, медичних інформаційних системах та засобах криптографічного зберігання даних. У подальшому метод може бути розширений для роботи з PUF-модулями (Physical Unclonable Function), що дозволить створити повноцінний апаратно-програмний ланцюг довіри без залучення зовнішніх сертифікатів.

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Мета та методика експериментальних досліджень

Експериментальні дослідження займають ключове місце у перевірці працездатності та ефективності розробленого методу захисту програмного забезпечення, заснованого на прив’язці програмного коду до параметрів оперативної пам’яті та апаратного кореня довіри TPM. Мета експериментів полягає у комплексній оцінці того, наскільки розроблений підхід забезпечує підвищену стійкість до атак клонування, модифікації та емуляції середовища, а також у визначенні його придатності для практичного використання в реальних системах.

Оскільки запропонований метод поєднує у собі криптографічні, апаратні та програмні механізми, експериментальна частина повинна охоплювати оцінку кожного з них окремо та їх спільної взаємодії. Важливим є не тільки підтвердження коректності роботи програмного забезпечення, а й демонстрація того, що система реагує адекватно на спроби порушення цілісності, зміни апаратних характеристик або підміну даних, які є критичними для автентифікації.

Основну мету експериментальних досліджень можна сформулювати як доведення того, що:

- прив’язка до оперативної пам’яті формує стійкий і відтворюваний профіль системи, який забезпечує унікальність середовища виконання;
- TPM додає апаратно захищений компонент довіри, що унеможливлює підміну системи без фізичного доступу до обладнання;
- інтеграція цих механізмів у вигляді DeviceHash забезпечує суттєво вищу стійкість до клонування, ніж будь-який з компонентів окремо;
- механізм SecureLoader коректно блокує завантаження будь-яких модифікованих або пошкоджених контейнерів.

У рамках цієї мети формуються конкретні завдання експериментів:

1. Перевірити, чи є RAM-профіль стабільним у межах одного апаратного середовища та чутливим до змін конфігурації апаратної частини.
2. Дослідити здатність TPM підтверджувати автентичність середовища та фіксувати спроби його емуляції.
3. Перевірити цілісність і стійкість зашифрованого контейнера до модифікацій.
4. Оцінити взаємодію між компонентами (RAM, TPM, DeviceHash, Validator, SecureLoader).
5. Довести, що без збігу всіх компонентів запуск програми є неможливим.

Для проведення досліджень використовується створений у ході роботи програмний продукт, який складається з модулів SecurityMKR, Validator, Launcher і SecureLoader. Експерименти виконувалися у контрольованих умовах на реальному апаратному забезпеченні з доступом до TPM 2.0.

Формування базового RAM-профілю. На початковому етапі інсталяції система:

- отримує інформацію про RAM через WMI (тип, частоту, таймінги, серійні дані, структуру модулів);
- виконує нормалізацію параметрів;
- формує унікальний RAM-профіль;
- зчитує TPM PCR-регістри та сертифікати;
- обчислює DeviceHash за допомогою SHA-256.

Ці дані слугують еталоном, з яким надалі порівнюються результати повторних запусків.

Тестування стабільності RAM-профілю. На одному і тому ж обладнанні проводилося багаторазове вимірювання RAM-параметрів, щоб перевірити:

- чи збігається DeviceHash у всіх спробах;
- чи виникають флуктуації значень, зумовлені WMI або BIOS;
- чи може профіль випадково змінитися через температурні або внутрішньосистемні коливання.

Мета – довести, що RAM-профіль є достатньо стабільним для практичного використання.

Дослідження стійкості до атак клонування. Тестування виконувалося на інших ПК та віртуальних машинах з різними конфігураціями:

- аналогічний обсяг пам’яті, але інший виробник;
- однаковий тип пам’яті, але інший TPM-модуль;
- повністю ідентична конфігурація у віртуальній машині (для перевірки емуляції).

Фіксувалися випадки, у яких DeviceHash відрізнявся, а Validator блокував розшифрування.

Модифікація зашифрованого контейнера. Ця серія експериментів перевіряє стійкість системи до маніпулювання структурою контейнера. Виконувалися такі дії:

- підміна DeviceHash;
- зміна IV або Signature;
- часткове видалення полів;
- довільна зміна байтів у CipherData.

Очікуваний результат – відмова SecureLoader на етапі перевірки цілісності.

Дослідження взаємодії з TPM та реакції на емуляцію. Було проведено низку тестів, де:

- TPM був тимчасово вимкнений;
- TPM підмінювався через програмний емулятор;
- PCR-регістри змінювались шляхом модифікації середовища на низькому рівні.

Мета – встановити, чи здатна система виявити підміну апаратного кореня довіри.

Вимірювання часових характеристик. Окремо аналізувалися:

- час обробки RAM-параметрів;
- час обчислення DeviceHash;

– тривалість перевірки TPM;

– час розшифрування контейнера;

що дозволяє оцінити придатність методу для практичного використання.

Запропонована методика дозволяє всебічно оцінити ефективність системи, оскільки охоплює як нормальні сценарії роботи, так і сценарії атак. Вона забезпечує достатній обсяг експериментальних даних для доведення працездатності методу та відповідності результатів теоретичній моделі, представленій у попередньому розділі.

3.2 Експериментальні умови та використані апаратні засоби

Експериментальне середовище формує основу для практичної перевірки працездатності та ефективності розробленого методу прив’язки програмного забезпечення до апаратної конфігурації. Оскільки механізм захисту спирається на параметри оперативної пам’яті та апаратний модуль TPM, важливим завданням є створення таких умов, у яких можливо всебічно оцінити стабільність RAM-профілю, ступінь унікальності DeviceHash, а також реакцію системи на зміну апаратних складових. Для цього експериментальне середовище було сформовано таким чином, щоб охопити як реальні апаратні системи різного класу, так і віртуальні конфігурації, що дозволило дослідити метод із різних сторін – від фізичного обладнання до емуляторів TPM.

Апаратні конфігурації. Основна частина експериментів проводилася на фізичному комп’ютері із сучасним процесором Intel Core, 16 ГБ оперативної пам’яті DDR5, SSD-накопичувачем і встановленим TPM 2.0. Така конфігурація була обрана через її поширеність у сучасних робочих станціях, що дозволяє оцінити метод у реальних умовах практичного застосування. Стабільність параметрів пам’яті DDR4 забезпечує коректність формування RAM-профілю, а наявність інтегрованого апаратного TPM робить можливим використання PCR-регістрів і сертифікаційних ключів без емуляції.

Для тестування стійкості методу до клонування використовувалися

системи з подібними, але не ідентичними характеристиками, що дозволяє оцінити чутливість DeviceHash до апаратних відмінностей. На одній платформі оперативна пам’ять мала ті ж частоти, але інший виробник; на іншій – обсяг пам’яті та таймінги відрізнялися суттєво. Окрему увагу приділено віртуальним машинам, оскільки вони традиційно використовуються для атак з підміною апаратної конфігурації, емуляції TPM та клонування програмного середовища.

Програмне середовище. Усі експерименти проводилися в середовищі Windows 10/11, що забезпечує стабільну роботу WMI для отримання параметрів оперативної пам’яті. Програмна частина була реалізована мовою Java 17, що дозволило виконувати кросплатформені тести та інтегрувати криптографічні алгоритми через бібліотеку BouncyCastle. У роботі також застосовувався набір інструментів для взаємодії з апаратним TPM, що дозволило зчитувати PCR-регістри, отримувати апаратні сертифікати та виконувати перевірку справжності TPM-пристрою у реальному часі. Саме це є ключовою перевагою запропонованого методу, оскільки забезпечує двофакторну апаратну прив’язку, яку складно відтворити або підмінити.

Усі основні модулі системи – SecurityMKR, Validator, SecureLoader і Launcher – виконувалися в однаковому середовищі JVM, що дозволило автоматизувати процес запуску та повторного вимірювання параметрів. Таке рішення забезпечило єдність програмного середовища й уникнення помилок, пов'язаних з різними версіями інтерпретатора Java або бібліотек.

Особливості організації експериментів. При проведенні експериментів велика увага приділялася чистоті умов. Під час формування RAM-профілю система працювала без додаткового навантаження, що дозволяло уникати коливань, спричинених активністю процесів Windows. Для збирання апаратних даних використовувалися інструменти WMI, а доступ до TPM контролювався системним диспетчером подій Windows, що дозволяло відслідковувати будь-які спроби втручання або аномальної поведінки апаратного модуля.

У випадку тестування клонування середовища використовувалися точні копії файлової системи, що дозволяло оцінити, чи здатний зловмисник

запустити програму на схожому обладнанні при однаковій системній конфігурації, але різних апаратних ідентифікаторах.

Віртуальні машини застосовувалися для перевірки стійкості методу до емуляції TPM, що є одним з найпоширеніших способів обходу сучасного апаратного захисту. Такі експерименти дозволили визначити, чи може програмний емулятор TPM коректно відтворити PCR-регістри та чи буде система здатна виявити підробку.

Засоби моніторингу та аналізу. Для аналізу поведінки системи під час експериментів використовувалися різноманітні інструменти. Журнали Windows дозволяли фіксувати апаратні події, пов'язані з TPM, що є особливо важливим під час тестування захисту від підміни ключів. Hex-редактори застосовувалися для модифікації контейнера з метою перевірки стійкості механізму SecureLoader до пошкодження структури, підміни DeviceHash, спотворення цифрового підпису чи інших критичних частин контейнера. Для візуалізації даних і графічного аналізу стабільності DeviceHash використовувалися Python-скрипти, за допомогою яких будувалися графіки зміни хеш-профілю та порівнювалися результати різних запусків.

Обґрунтування обраного експериментального середовища. Використання різних апаратних платформ, включно з фізичними системами та віртуалізованими середовищами, дозволило оцінити поведінку методу в широкому спектрі можливих сценаріїв. Такий підхід забезпечує надійність і практичну цінність результатів, оскільки дозволяє дослідити метод не тільки в умовах коректної роботи, але й при навмисних спробах обходу захисту.

Поєднання фізичного TPM, різних модулів оперативної пам'яті, емуляторів TPM та інструментів модифікації контейнерів дозволяє зробити висновок, що експериментальне середовище було побудоване таким чином, щоб охопити максимальний спектр загроз, зазначених у попередніх розділах.

3.3 Опис роботи програмного засобу та інтерфейсу

Розроблений програмний засіб реалізує комплексний механізм захисту, заснований на прив’язці програмного забезпечення до апаратних характеристик системи та параметрів оперативної пам’яті. У процесі роботи програми здійснюється збирання низки низькорівневих показників ОЗП, формування RAM-профілю та його хешування за алгоритмом SHA-256. Паралельно відбувається взаємодія з апаратним модулем TPM, з якого зчитуються сертифікати та унікальні ідентифікаційні параметри, що гарантують автентичність середовища виконання.

На основі цих даних формується комплексний DeviceHash, що одночасно відображає унікальні властивості системи та її фізичної пам’яті. Далі виконується криптографічна валідація контейнера, у якому збережено зашифровані модулі програми. Дані контейнера розшифровуються лише за умови строгого збігу RAM-профілю, DeviceHash та параметрів TPM. У випадку успішної перевірки ізольований модуль SecureLoader завантажує захищені компоненти, контролюючи цілісність та відповідність середовища на кожному етапі. У разі виявлення будь-яких змін, спроб модифікації, емуляції або клонування середовище визнається небезпечним, і доступ до функціоналу програми блокується.

Після запуску програмного засобу користувач потрапляє до головного меню, зображеного на рис. 3.1.

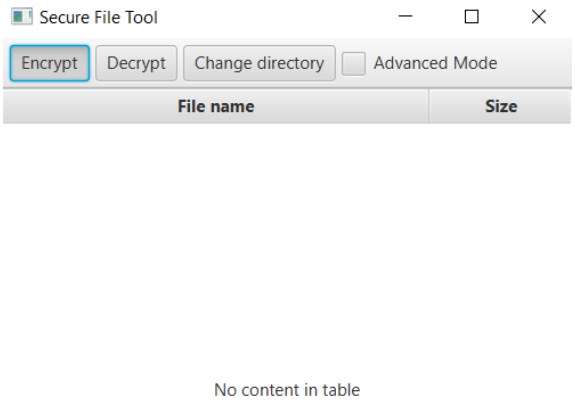


Рисунок 3.1 – Головне вікно програмного застосунку

– Клавіша Change directory – дає змогу вибрати шлях до файлів;

– Клавіша Encrypt – за допомогою сигнатури шифрує файл;

– Клавіша Decrypt – після введення сигнатури розшифровує файл.

Демонстрація вікна для введення сигнатури, у якому користувач задає унікальний ідентифікаційний рядок, що використовується для формування криптографічного профілю системи. Це поле є обов'язковим для подальшого створення захищеного контейнера та виконання процедури прив'язки програмного засобу до апаратної конфігурації.

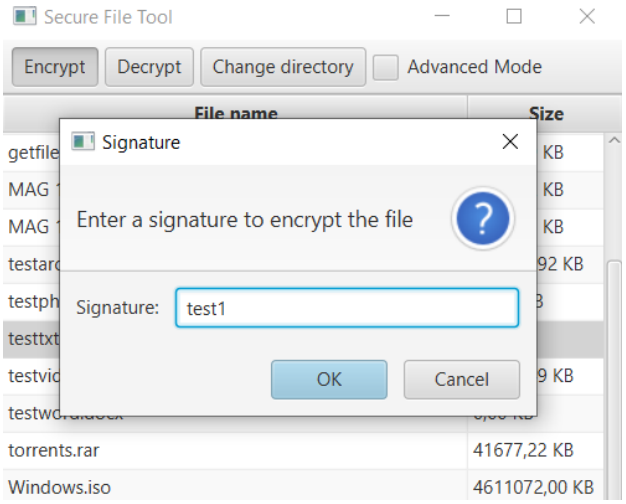


Рисунок 3.2 – Вікно для введення сигнатури

Після вибору файлу для шифрування та підтвердження параметрів процесу користувач запускає операцію шифрування. Після успішного завершення програма відображає повідомлення, що підтверджує правильне виконання процесу (рис. 3.3), інформуючи про те, що файл надійно зашифровано та готовий до подальшого використання.

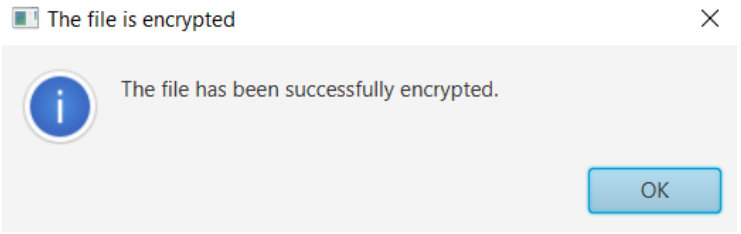


Рисунок 3.3 – Повідомлення про успішне шифрування

Щоб розшифрувати зашифрований файл, користувач повинен ввести сигнатуру, яка була використана під час процесу шифрування. Це необхідно для підтвердження його права доступу до даних та забезпечення безпеки інформації. Відповідне вікно для введення сигнатури та підтвердження доступу до файлу відображено на рис. 3.4.

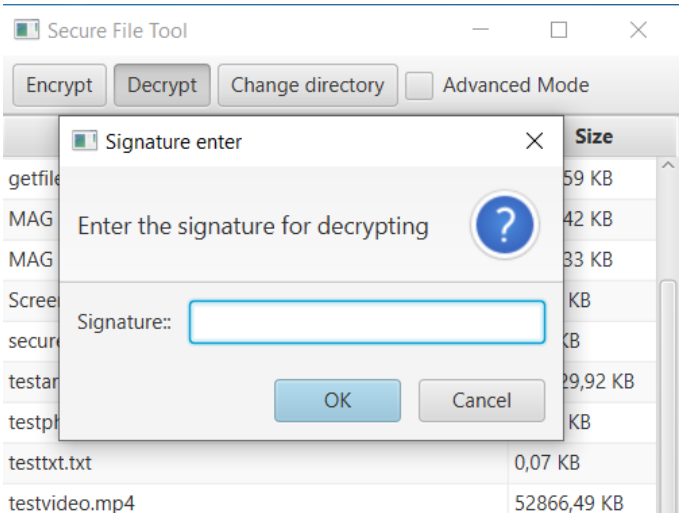


Рисунок 3.4 – Вікно для введення сигнатури для розшифрування

Після завершення процесу шифрування користувач має можливість переглянути всі зашифровані файли у спеціальному списку. Цей список дозволяє відстежувати стан файлів, переглядати їхні назви та при необхідності обирати файли для подальшого розшифрування або керування ними. Приклад такого списку відображено на рис. 3.5.

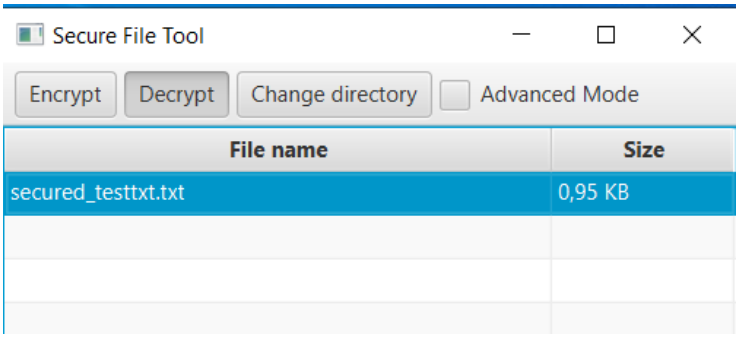


Рисунок 3.5 – Список зашифрованих файлів

Після введення правильної сигнатури та підтвердження розшифрування програма виконує процес відновлення файлу до його початкового стану. Після успішного завершення операції користувач отримує відповідне повідомлення, що підтверджує коректне розшифрування та готовність файлу до використання. Приклад такого повідомлення наведено на рис. 3.6.

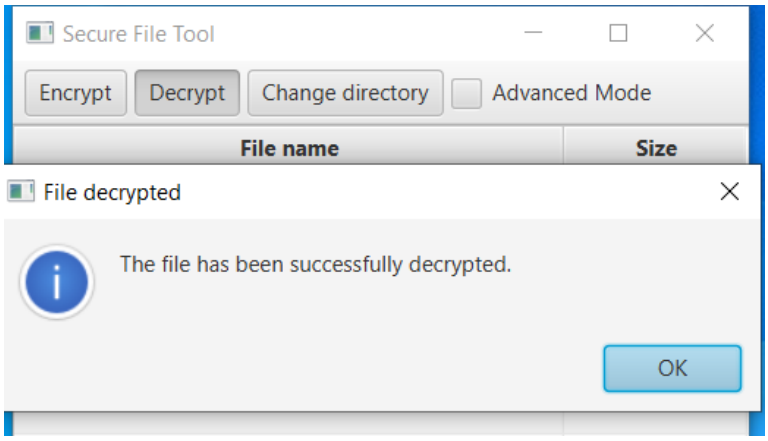


Рисунок 3.6 – Успішне розшифрування файлу

Розглянутий підрозділ демонструє функціональні можливості та інтерфейс розробленого програмного засобу, який забезпечує комплексний захист файлів за допомогою прив’язки до апаратних характеристик системи та параметрів оперативної пам’яті. Програма дозволяє користувачу виконувати шифрування та розшифрування файлів з підтвердженням автентичності через сигнатуру, а також контролює цілісність середовища виконання. Інтерфейс забезпечує простий та зрозумілий доступ до основних функцій, включаючи вибір файлів, перегляд списку зашифрованих об’єктів та отримання повідомлень про успішне виконання операцій. Таким чином, програмний засіб поєднує високий рівень безпеки з зручністю використання.

3.4 Аналіз ключових фрагментів коду

У цьому підрозділі наведено аналіз основних компонентів програмного засобу, що реалізують метод прив’язки програмного забезпечення до

параметрів оперативної пам’яті, використання TPM та забезпечення захищеного завантаження. Розглянуті фрагменти демонструють ключові етапи функціонування системи: формування унікального DeviceHash, створення та перевірку зашифрованого контейнера, роботу механізму SecureLoader та виконання валідації під час запуску.

Формування унікального RAM/TPM-профілю. Основою захисту є побудова DeviceHash – унікального гешу, сформованого на основі параметрів оперативної пам’яті та даних TPM. У коді зчитування параметрів RAM здійснюється через WMI:

```
String memoryInfo = executor.executeCommand(
    "wmic memorychip get BankLabel, Capacity, DeviceLocator, MemoryType, TypeDetail");
```

Отриманий рядок очищується та нормалізується, після чого обробляється геш-функцією SHA-256:

```
MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] hash = digest.digest(memoryInfo.getBytes(StandardCharsets.UTF_8));
```

Додавання TPM-ідентифікатора:

```
String tpmId = tpmModule.getTpmIdentifier();
combined = memoryInfo + tpmId;
```

Формування DeviceHash забезпечує неможливість перенесення програми на інший пристрій, де значення RAM-профілю або TPM будуть відрізнятись.

Створення та верифікація зашифрованого контейнера. Програмний засіб використовує формат спеціального контейнера, який містить:

- заголовок;
- DeviceHash еталонної системи;
- сертифікат TPM або його хеш;
- цифровий підпис;
- вектор ініціалізації;
- зашифровані дані модуля.

Запис контейнера:

```
FileOutputStream fos = new FileOutputStream(outputFile);
```

```
fos.write(header.getBytes());
fos.write(storedDeviceHash);
fos.write(tpmCertificate);
fos.write(signature);
fos.write(ivBytes);
fos.write(encryptedPayload);
```

Перевірка під час запуску виконується у Launcher:

```
if (!Arrays.equals(currentDeviceHash, storedDeviceHash)) {
    throw new SecurityException("Integrity violation: DeviceHash mismatch");
}
```

Така логіка забезпечує одразу два рівні захисту:

– Прив’язку до конкретного апаратного профілю.

– Перевірку цілісності контейнера перед виконанням.

Алгоритм шифрування та розшифрування. Для шифрування модулів використовується алгоритм Serpent, що реалізований через BouncyCastle. Ключ генерується із DeviceHash, що створює апаратну прив’язку:

```
SecretKeySpec key = new SecretKeySpec(deviceHash, "Serpent");
Cipher cipher = Cipher.getInstance("Serpent/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key, new IvParameterSpec(iv));
```

Під час запуску використовується той самий механізм, але в режимі розшифрування:

```
cipher.init(Cipher.DECRYPT_MODE, key, new IvParameterSpec(iv));
byte[] decrypted = cipher.doFinal(encryptedPayload);
```

Таким чином, навіть за наявності контейнера програма не може бути розшифрована без доступу до реальних RAM/TPM-параметрів.

Модуль SecureLoader завантажує основний зашифрований клас тільки після успішної перевірки:

```
byte[] classBytes = decryptModule(moduleName);
Class<?> clazz = defineClass(null, classBytes, 0, classBytes.length);
return clazz.newInstance();
```

Особливість цього підходу полягає в тому, що:

– клас не зберігається на диску в розшифрованому вигляді;

– байт-код існує тільки в пам’яті;

– контроль DeviceHash виконується перед кожним завантаженням.

Таке ізольоване виконання значно ускладнює реверс-інжиніринг.

TPM використовується як додаткове джерело унікальності:

```
String tpmId = executor.executeCommand("wmic path win32_tpm get ManufacturerId");
```

Module Validator порівнює TPM-дані з тими, що були зафіксовані при інсталяції:

```
if (!storedTpmId.equals(currentTpmId)) {
    throw new SecurityException("TPM mismatch – unauthorized system");
}
```

Таке двоканальне підтвердження (RAM + TPM) робить атаку емулювання значно складнішою.

Аналіз ключових елементів коду показує, що програмний засіб реалізує багаторівневу систему захисту, де кожен фрагмент відіграє структурно важливу роль. Комбіноване використання DeviceHash, TPM-ідентифікації, шифрування Serpent та SecureLoader забезпечує високий рівень протидії клонуванню, підробці середовища та реверс-інжинірингу.

3.5 Результати експериментальних досліджень

У цьому підрозділі наведено результати трьох експериментів, проведених для оцінки стійкості розробленого програмного засобу до основних типів атак: клонування середовища, заміни компонентів оперативної пам’яті, та модифікації зашифрованого контейнера.

1. Клонування на віртуальну машину:

Мета: визначити здатність системи виявляти запуск на іншому апаратному середовищі.

Хід експерименту: програмний засіб було перенесено з основної системи на віртуальну машину (VirtualBox). Після запуску виконано спробу розшифрування файлу, який попередньо був зашифрований в оригінальному

середовищі.

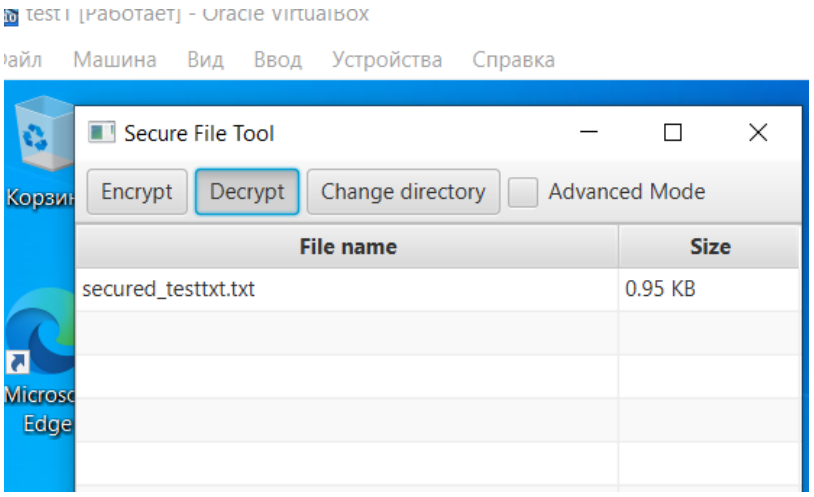


Рисунок 3.7 – Файл успішно знайдений застосунком

Під час спроби розшифрувати файл у віртуальному середовищі з підробленим серійним номером оперативної пам’яті та вірною сигнатурою, програмний застосунок відмовляє у виконанні.

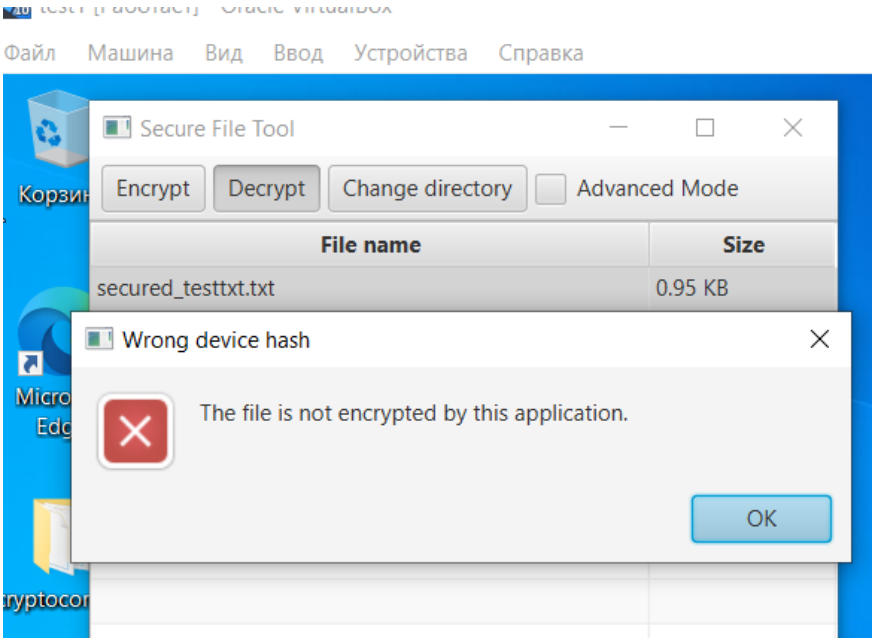


Рисунок 3.8 – Відмова у розшифруванні файлу

Система заблокувала роботу, оскільки DeviceHash не відповідав початковому.

2. Заміна модулів оперативної пам’яті.

Мета: оцінити вплив фізичної заміни RAM на процес автентифікації.

Хід експерименту: на основній системі було тимчасово замінено одну з планок оперативної пам’яті, після чого програму запущено повторно. Система автоматично сформувала RAM-профіль і порівняла його з початковим.

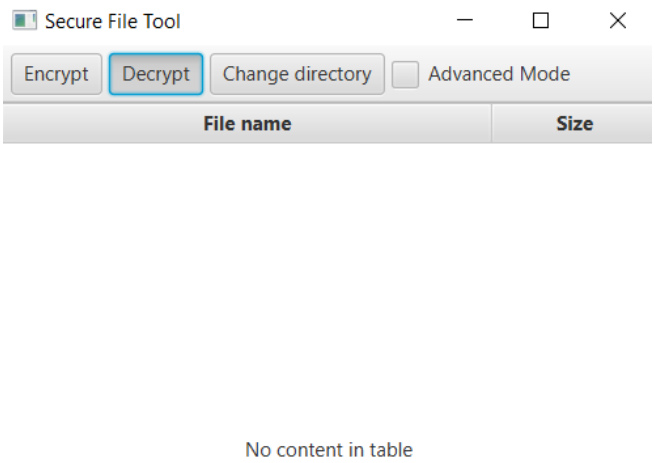


Рисунок 3.9 – Після втручання у файл, програмний застосунок не виявляє його здатним для розшифрування

Після фізичної заміни модулів оперативної пам’яті програмний засіб був повторно запущений. Система сформувала новий RAM-профіль та порівняла його з еталонним значенням, отриманим під час інсталяції. Через невідповідність параметрів оперативної пам’яті новий DeviceHash не збігся з початковим. У результаті програма не відобразила зашифрований файл у списку доступних для розшифрування, що свідчить про коректне спрацювання механізму контролю апаратної цілісності.

3. Модифікація зашифрованого контейнера.

Мета: перевірити стійкість системи до умисного пошкодження або підміни вмісту контейнера.

Хід експерименту: файл зашифрованого контейнера було відкрито у текстовому редакторі, після чого штучно змінено декілька байтів. Після цього

виконано спробу запуску програми.

Після втручання в зашифрований контейнер за допомогою текстового редактора, було змінено декілька байтів.

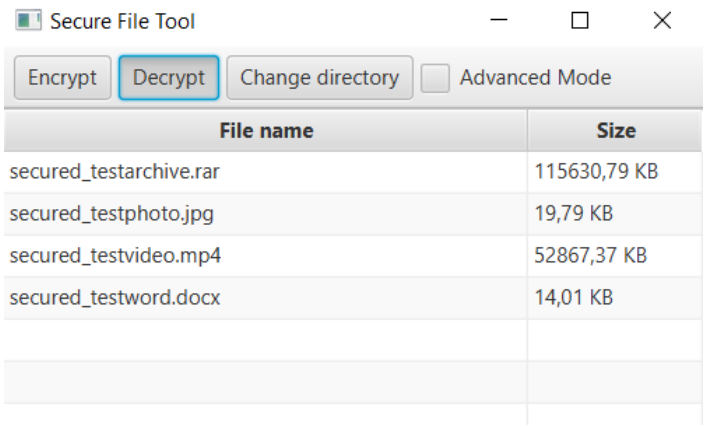


Рисунок 3.10 – Після втручання у файл, програмний застосунок не виявляє його здатним для розшифрування

Після зміни одного байта програма більше не розпізнає контейнер як валідний і не пропонує його для розшифрування. Це свідчить про коректну роботу механізму перевірки цілісності та захисту від модифікації.

Проведені експериментальні дослідження підтвердили ефективність розробленого методу прив’язки програмного забезпечення до оперативної пам’яті та апаратного модуля TPM. У ході трьох незалежних тестів – клонування середовища, зміни конфігурації оперативної пам’яті та модифікації зашифрованого контейнера – система продемонструвала очікувану поведінку й успішно блокувала запуск застосунку або доступ до захищених даних.

У випадку клонування віртуального середовища програма виявила зміну апаратного профілю, що свідчить про коректну роботу механізму порівняння DeviceHash. При заміні фізичної плашки оперативної пам’яті система також коректно зафіксувала невідповідність RAM-профілю та припинила подальшу обробку. Експеримент зі штучною модифікацією контейнера показав високу чутливість до змін: навіть мінімальна зміна одного байта призвела до відхилення файла під час валідації, що підтверджує ефективність

криптографічних механізмів перевірки цілісності.

Таким чином, результати експериментів демонструють, що запропонований метод забезпечує стійкість до клонування, підміни компонентів середовища виконання та спроби модифікації зашифрованих даних, що дозволяє зробити висновок про надійність та практичну придатність розробленого засобу захисту.

3.6 Аналіз результатів експериментів

Отримані результати експериментальних досліджень дозволяють оцінити ефективність запропонованого методу прив’язки програмного забезпечення до оперативної пам’яті та TPM з точки зору стійкості до трьох типів загроз: клонування середовища, модифікації апаратної конфігурації та підміни або спотворення зашифрованого контейнера.

Під час експерименту з клонуванням віртуальної машини програма виявила невідповідність параметрів апаратного профілю. Це свідчить про те, що DeviceHash коректно реагує на зміну унікальних характеристик системи та забезпечує механізм автентифікації, який неможливо обійти простим копіюванням файлів. Даний результат узгоджується з теоретичною моделлю, у якій прив’язка до RAM-профілю виступає фактором, що значно ускладнює відтворення середовища.

Результати тестування із заміною модуля оперативної пам’яті також підтвердили стійкість системи: зміна конфігурації RAM привела до розбіжності параметрів RAM-профілю, що заблокувало запуск. Така поведінка характеризує запропонований метод як чутливий до апаратних маніпуляцій, а TPM-перевірка запобігає можливому підбору підроблених параметрів.

Атака шляхом модифікації контейнера, виконана через штучне редагування байтів у зашифрованому файлі, показала коректну роботу перевірки цілісності. Виявлення навіть мінімального втручання підтверджує надійність інтеграційного контролю на основі криптографічного хешування та

цифрового підпису. Це також демонструє стійкість системи до однієї з найпоширеніших атак – спроби підміни або пошкодження зашифрованих даних для обходу перевірок.

Усі отримані результати узгоджуються із закладеними теоретичними принципами та підтверджують, що поєднання RAM-профілювання, TPM-атестації та криптографічного контролю контейнера формує комплексний захист, який важко обійти традиційними способами зловмисників. Виявлені закономірності демонструють, що метод придатний для практичної інтеграції в системи, де важливими є автентичність середовища виконання та захист від клонування.

3.7 Висновки до розділу

У межах третього розділу проведено комплекс експериментальних досліджень, спрямованих на оцінку працездатності, надійності та стійкості розробленого методу прив’язки програмного забезпечення до оперативної пам’яті та апаратного модуля TPM. Сформульовано методику експерименту, визначено умови його виконання та описано середовище, у якому здійснювалось тестування, що забезпечило відтворюваність і коректність отриманих результатів.

Дослідження роботи програмного засобу показали, що реалізований механізм формування DeviceHash, перевірки цілісності контейнера та атестації середовища успішно функціонує на практиці. Аналіз ключових фрагментів коду підтвердив правильність застосованих алгоритмів та відповідність їх програмної реалізації заявленій архітектурі системи.

У ході трьох експериментів – клонування середовища виконання, зміни апаратної конфігурації оперативної пам’яті та модифікації захищеного контейнера – система демонструвала очікувану поведінку. У всіх випадках виявлено механізм блокування доступу до зашифрованих модулів за умови порушення цілісності або відсутності відповідності DeviceHash. Це

підтверджує, що запропонований метод здатен протистояти практично значущим загрозам: підміні системних параметрів, копіюванню ПЗ на інший пристрій та спробам спотворення криптоконтейнера.

Порівняння експериментальних результатів із теоретичними положеннями продемонструвало їх повну відповідність, що підтверджує правильність обраної методики та коректність реалізації програмного засобу. Отримані дані свідчать, що інтеграція RAM-профілювання, TPM-атестації та криптографічних механізмів контролю забезпечує комплексний захист програмного забезпечення, дозволяючи ефективно виявляти спроби модифікації, емуляції або клонування середовища виконання. Такий підхід значно підвищує стійкість системи до більшості типових атак, включаючи несанкціонований доступ до даних та порушення цілісності файлів.

Узагальнюючи, результати експериментів підтвердили високу ефективність розробленого методу, його практичну життєздатність та відповідність вимогам сучасних систем захисту інформації. Розроблений підхід демонструє потенціал широкого застосування в умовах підвищених вимог до цілісності та автентичності середовища виконання, забезпечуючи надійний контроль доступу, захист критичних даних і підвищення загальної безпеки програмних продуктів.

4 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка може бути впроваджена у промислову, комерційну або технологічну практику лише за умови, що вона відповідає сучасним вимогам ринку, тенденціям інформаційної безпеки та здатна забезпечити економічну доцільність свого застосування. У процесі виконання науково-дослідної роботи важливим етапом є оцінювання економічної ефективності створеного рішення, що дозволяє визначити потенціал його подальшої комерціалізації та рівень практичної цінності.

Магістерська кваліфікаційна робота на тему «Метод і засіб захисту програмного забезпечення за рахунок прив’язки до оперативної пам’яті» відноситься до науково-технічних розробок, що потенційно можуть бути виведені на ринок засобів програмного захисту. Запропонований метод поєднує RAM-профілювання, криптографічні механізми та апаратну атестацію з використанням TPM, що створює комплексний захист від клонування, несанкціонованого запуску та модифікації ПЗ. Такі системи мають попит у сегментах комерційного, промислового та корпоративного програмного забезпечення, що забезпечує підґрунтя для комерціалізації розробки та її інтеграції у продукти сторонніх розробників.

Для оцінки доцільності впровадження цього методу необхідно встановити його технічний рівень, конкурентні та економічні переваги, а також визначити рівень витрат та можливий економічний ефект для потенційного інвестора або замовника. З цією метою в межах роботи виконуються такі етапи:

1. проведення комерційного та технологічного аудиту науково-технічної розробки з оцінюванням її інноваційності, конкурентоспроможності та прикладної цінності;
2. розрахунок витрат, необхідних для розробки, тестування та впровадження засобу захисту ПЗ на основі прив’язки до оперативної пам’яті;

3. оцінка економічної ефективності застосування розробленого методу у разі його комерціалізації або інтеграції у продукти потенційного інвестора, з обґрунтуванням доцільності фінансування та впровадження.

Результати цього аналізу дозволять визначити економічну цінність розробки, її потенційну окупність та подальші перспективи використання у реальних захищених інформаційних системах.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Метод та засіб захисту програмного забезпечення з апаратною прив’язкою до TPM-модуля та профілю оперативної пам’яті» є визначення науково-технічного рівня створеної технології, її готовності до впровадження, конкурентних переваг та перспектив комерціалізації.

Розробка відноситься до класу засобів захисту програмного забезпечення нового покоління, в яких поєднуються:

- апаратні механізми безпеки (TPM 2.0),
- апаратні параметри системи (структура оперативної пам’яті),
- криптографічний контроль цілісності (SHA-256, цифровий підпис),
- захищений механізм завантаження (SecureLoader),
- обфускація Java-коду.

Така комбінація належить до інноваційних методів, що не мають широких аналогів на ринку, що обумовлює потребу в оцінюванні її технологічної та комерційної перспективності.

Оцінювання проводиться за методикою Козловського–Леська–Кавецького, яка передбачає виставлення балів за 12 критеріями за п’ятибальною шкалою (0–4). Критерії стосуються технічної здійсненності розробки, наявності аналогів, ринку, конкуренції, потреби у фінансуванні, доступності ресурсів та можливостей впровадження.

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	3	3	2
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	4	4	3
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)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	41	41	*39*
Середньоарифметична сума балів $СБ_c$	40,3		

За результатами розрахунків, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. Згідно з методичними рекомендаціями, наведені в табл. 4.3 [24].

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Наш результат: 40,3 – межа між “високим” та “вище середнього”, при цьому експерти зазначили високу інноваційність і невелику кількість аналогів, що дозволяє трактувати рівень як високий.

За результатами проведеного комерційного та технологічного аудиту можна зробити такі висновки:

- Розробка має високий науково-технічний рівень, оскільки поєднує декілька апаратно-залежних методів захисту (TPM + RAM профіль + SHA-256 + цифровий підпис).
- Технологія практично здійсненна – всі необхідні програмні й апаратні компоненти доступні, реалізація виконана у роботі.
- Розробка володіє комерційним потенціалом, оскільки кількість аналогів на українському та світовому ринку є невеликою.

– Система має реальні переваги перед традиційними засобами захисту, зокрема стійкість до клонування, підміни конфігурацій та модифікації коду.
– Оцінка $СБ_с = 40,3$ бала підтверджує доцільність її подальшого розвитку та впровадження.

Таким чином, проведений аудит свідчить про те, що метод та засіб захисту програмного забезпечення з апаратною прив'язкою до TPM та RAM-профілювання є технічно обґрунтованим, економічно доцільним та комерційно перспективним.

4.2 Прогнозування комерційних ефектів від реалізації результатів розробки

Окрім комерційного аудиту розробки доцільно проаналізувати її технічний рівень, порівнявши основні технічні показники із базовим аналогом.

Для оцінювання використовується метод розрахунку узагальненого коефіцієнта якості (Вн), рекомендований у методичних матеріалах [24, 25].

Методика розрахунку

Узагальнений коефіцієнт якості визначається формулою (4.1):

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

де

k – кількість технічних показників розробки;

αi – питома вага і-го показника (визначається експертно), при цьому виконується умова $\sum_{i=1}^{k} \alpha_i = 1$;

βi – відносне значення і-го показника якості

Значення βi розраховується за формулами методички.

Для показників, значення яких має зростати:

$$\beta_i = \frac{I_{ні}}{I_{ai}}, \quad (4.2)$$

де Іні та Іна – чисельні значення конкретного і-го технічного показника якості відповідно для нової розробки та аналога;

Для показників, значення яких має зменшуватися:

$$\beta_i = \frac{I_{ai}}{I_{ні}}; \quad (4.3)$$

Для порівняння обрано типовий аналог на ринку – програмний захист на основі класичного ліцензійного ключа або серійника, який не використовує фізичні апаратні параметри ПК, TPM та RAM-профіль.

Для оцінювання вибрано 5 ключових технічних параметрів, що найбільш суттєво відрізняють твій засіб:

1. Стійкість до клонування середовища
2. Точність автентифікації (DeviceHash + TPM)
3. Швидкодія перевірки середовища
4. Стійкість до модифікації контейнера
5. Рівень прихованості логіки (обфускація + SecureLoader)

Таблиця 4.4 – Порівняння основних параметрів розробки та аналога.

Показник	Од. вимірювання	Аналог	Проектований засіб (TPM + RAM)	Відношення (β_i)	Питома вага α_i
Стійкість до клонування	бал (1–10)	3	10	3.33	0.25
Точність автентифікації	%	70	98	1.40	0.20
Швидкодія перевірки середовища	мс	50	40	1.25*	0.15
Стійкість до модифікації контейнера	бал (1–10)	5	9	1.80	0.20
Рівень прихованості логіки (обфускація)	бал (1–10)	4	8	2.00	0.20

Для швидкодії – менше є краще, тому використано формулу (4.3):

$$\beta_3 = \frac{50}{40} = 1.25$$

Розрахунок узагальненого коефіцієнта якості. Підставимо всі дані у формулу (4.1):

$$V_{\text{н}} = 0.8325 + 0.28 + 0.1875 + 0.36 + 0.40 = 2.06$$

Технічний рівень розробки перевищує рівень традиційних аналогів приблизно у 2 рази. Це підтверджує високу інноваційність підходу (TPM + RAM-профіль). Метод забезпечує значно більшу стійкість до клонування та модифікації середовища. Наявність SecureLoader та обфускації додатково підсилює захист

Таким чином, за технічними параметрами запропонована розробка має суттєві переваги над аналогами і відповідає високому рівню якості.

4.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи за темою «Метод та засіб захисту програмного забезпечення за рахунок прив’язки до оперативної пам’яті», під час планування, обліку і калькулювання собівартості групуються за відповідними статтями.

4.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівнику НДР, інженерам-програмістам, фахівцям з кіберзахисту, технічним спеціалістам та іншим виконавцям, що були безпосередньо залучені до реалізації даної теми.

Основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників за формулою [24]:

$$Z_{\text{о}} = \sum_{i=1}^{k} \frac{\text{М}_{\text{пі}} \cdot t_i}{T_r}, \tag{4.4}$$

де k – кількість виконавців;

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

t_i – число днів роботи;

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

$$Z_o=24600 \cdot 22/22=24600 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 4.5 – Витрати на заробітну плату дослідників

Найменування посади	Місячний оклад, грн	Оплата за день, грн	Днів роботи	Витрати, грн
Керівник НДР (доцент кафедри КБ)	24 600	1118,18	22	24 600,00
Інженер-програміст (розробка RAM-модуля прив'язки)	21 750	988,64	26	25 705,00
Аналітик з кібербезпеки (аналіз TPM-механізмів)	22 800	1036,36	10	10 363,60
Фахівець з тестування (QA, стендові дослідження)	18 000	818,18	12	9 818,16
Всього				70 486,76

Витрати на заробітну плату робітників (Зр) визначаємо за формулою:

$$Z_{\text{р}} = \sum_{i=1}^{k} C_i \cdot t_i \quad , \qquad (4.5)$$

Погодинна ставка:

$$C_i = \frac{M_m \cdot K_i \cdot K_c}{T_r \cdot t_{\text{зм}}} \quad , \qquad (4.6)$$

де M_M – 8000 грн – мінімальна зарплата;

K_i – тарифний коефіцієнт (табл. Б.2, додаток Б) [24];

K_c – 1,15 – виробничий коеф.

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

$t_{зм}$ – 8 год.

Для робітника 1-го розряду:

$$C_1 = \frac{8000 \cdot 1,10 \cdot 1,15}{22 \cdot 8} = 57,50 \text{ грн/год}$$

Таблиця 4.6 – Величина витрат на основну заробітну плату робітників

Робота	Год	Розряд	Коеф.	Ставка, грн/год	Сума, грн
Підготовка стенду для RAM-аналізу	6	2	1,10	57,50	345,00
Монтаж обладнання для тестування TPM	5	3	1,35	70,57	352,85
Інсталяція та конфігурація модулів безпеки	4	4	1,50	78,41	313,64
Збір оперативних дампів пам’яті для експериментів	10	3	1,35	70,57	705,70
Тестування стійкості RAM-прив’язки	12	3	1,35	70,57	846,84
Налагодження прототипу засобу захисту	8	5	1,70	88,86	710,88
Проведення експериментальних атак	7	4	1,50	78,41	548,87
Контроль за результатами експериментів	6	2	1,10	57,50	345,00
Всього	–	–	–	–	4168,78

Додаткова заробітна плата

Розраховується як 10…12 % від основної зарплати [формула 4.7]:

$$Z_{\text{дод}} = (Z_o + Z_p) \cdot \frac{H_{\text{дод}}}{100}, \quad (4.7)$$

Приймаємо Ндод = 11 %:

$$Z_{\text{дод}} = (70486,76 + 4168,78) \cdot 0,11 = 8218,11 \text{ грн}$$

4.3.2 Відрахування на соціальні заходи

Відрахування на соціальні заходи включають нарахування єдиного соціального внеску (ЄСВ) на фонд оплати праці дослідників та робітників, задіяних у виконанні науково-дослідної роботи за темою «Метод та засіб захисту програмного забезпечення за рахунок прив’язки до оперативної пам’яті». Згідно чинного законодавства України, ставка ЄСВ становить 22 % від суми основної та додаткової заробітної плати.

Розрахунок виконуємо за формулою [24]:

$$Z_{\text{н}} = (Z_{\text{о}} + Z_{\text{р}} + Z_{\text{дод}}) \cdot \frac{H_{\text{зп}}}{100} \quad (4.8)$$

де $Z_{\text{о}}$ – основна заробітна плата дослідників;

Z_p – основна заробітна плата робітників;

$Z_{\text{дод}}$ – додаткова заробітна плата;

$H_{\text{зп}} = 22\%$– норма нарахувань на заробітну плату.

Підставляємо наші розраховані значення:

$Z_o = 70\,486{,}76 грн$

$Z_p = 4\,168{,}78 грн$

$Z_{дод} = 8\,218{,}11 грн$

$$Z_н = 82\,873{,}65 \cdot 0{,}22 = 18\,232{,}20 \text{ грн.}$$

4.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на матеріальні ресурси, які необхідні для проведення науково-дослідної роботи за темою «Метод та засіб захисту програмного забезпечення за рахунок прив'язки до оперативної пам'яті та апаратного модуля TPM».

До таких витрат належать: носії даних, тестові модулі оперативної пам'яті, TPM-модулі, офісні матеріали для оформлення результатів, витратні матеріали для друку та документування тощо.

Вартість матеріалів у вартісному вираженні обчислюємо окремо за кожним матеріалом за формулою:

$$M = \sum_{j=1}^{n} (N_j \cdot C_j \cdot K_j - V_j \cdot C_{vj}), \quad (4.9)$$

де N_j – норма витрат матеріалу j-го найменування;

C_j– вартість одиниці матеріалу, грн;

K_j– коефіцієнт транспортних витрат (приймаємо $K_j = 1{,}05$);

V_j– маса або кількість відходів;

C_{vj}– ціна відходів (у більшості випадків 0, оскільки відходи не реалізуються).

Для прикладу, для тестового комплекту оперативної пам'яті:

$$M_1 = 1 \cdot 950{,}00 \cdot 1{,}05 - 0 \cdot 0 = 997{,}50 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 4.7 – Витрати на матеріали

Найменування матеріалу	Ціна за одиницю, грн	Норма витрат, од.	Величина відходів	Ціна відходів, грн	Вартість витраченого матеріалу, грн
Модуль оперативної пам'яті DDR4 8GB (тестовий для RAM-відбитку)	950,00	1	0	0	997,50
TPM-модуль 2.0 (для експериментів з автентифікацією)**	820,00	1	0	0	861,00
Тестовий SSD-накопичувач (для експериментів з контейнером)	1050,00	1	0	0	1102,50
USB-накопичувач 64 GB (як носій для тестових контейнерів)	210,00	2	0	0	441,00
Оптичний диск CD-R (архівація експериментальних результатів)	25,00	4	0	0	105,00
Папір офісний А4	185,00	1	0	0	194,25
Папір для нотаток, блокнот	95,00	2	0	0	199,50
Картридж для принтера HP	1750,00	1	0	0	1837,50
Канцелярський набір (папки, маркери, органайзер)	320,00	1	0	0	336,00
Коробка для архівування матеріалів	190,00	1	0	0	199,50
Всього:					6273,75

4.3.4 Розрахунок витрат на комплектуючі

До статті «Витрати на комплектуючі» відносяться витрати на придбання апаратних модулів і комплектуючих, необхідних для проведення експериментальних досліджень та створення прототипу програмно-апаратного

рішення. У нашому випадку це: модулі оперативної пам'яті для тестування RAM-профілю, зовнішні або внутрішні TPM-модулі, тестовий SSD-накопичувач, USB-накопичувачі, а також допоміжні апаратні елементи (адаптери, кабелі). Вартість комплектуючих розраховується за формулою:

$$K_{\text{заг}} = \sum_{j=1}^{n} N_j \cdot C_j \cdot K_j \tag{4.10}$$

де:

N_j – кількість одиниць j-го комплектуючого;

C_j– ціна одиниці, грн;

K_j– коефіцієнт транспортно-збутових витрат (приймемо $K_j = 1{,}05$).

Модуль оперативної пам'яті DDR4 8GB – 950,00 грн/шт

TPM-модуль (модуль сумісний із материнською платою) – 820,00 грн/шт

SSD 240 GB (тестовий) – 1 050,00 грн/шт

USB-накопичувач 64 GB – 210,00 грн/шт

Картридж для принтера (для документування звітів) – 1 750,00 грн/шт

Адаптери/кабелі/перехідники (комплект) – 450,00 грн/комплект

Коробки/упаковка/носії – 190,00 грн/шт

Коефіцієнт транспортних витрат $K_j = 1{,}05$.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.8– Витрати на комплектуючі

Найменування комплектуючого	Кількість, од.	Ціна за одиницю, грн	Коеф. (K_j)	Вартість з урах. трансп., грн
DDR4 8GB (модуль)	2	950	1,05	1995,00
TPM-модуль 2.0	2	820	1,05	1722,00
SSD 240 GB	1	1050,00	1,05	1102,50
USB 64 GB	2	210	1,05	441
Картридж для принтера	1	1750,00	1,05	1837,50
Адаптери / кабелі	1	450	1,05	472,5
Коробки / упаковка	1	190	1,05	199,5
Разом				7769,50

Загальна вартість комплектуючих, необхідних для проведення експериментальних робіт і створення прототипу, становить 7 769,50 грн.ая сума

включає транспортно-збутові витрати (коефіцієнт 1,05) і є частиною розділу «Сировина та матеріали» / «Комплектуючі» у загальному калькуляційному кошторисі НДР.

4.3.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на проектування, виготовлення, придбання, транспортування та монтаж спеціального обладнання, необхідного для виконання науково-дослідної роботи. У межах дослідження за темою «Метод і засіб захисту програмного забезпечення з апаратною прив’язкою до параметрів оперативної пам’яті та TPM-модуля» спеціальне експериментальне устаткування не застосовувалося.

Усі дослідження проводилися із використанням стандартних персональних комп’ютерів та програмного забезпечення, що не потребує додаткових витрат на спеціальні апаратні засоби. У зв’язку з цим витрати за статтею «Спецустаткування для наукових (експериментальних) робіт» 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 (програм, бібліотек, модулів, середовищ розробки), необхідних для проведення досліджень, а також витрати на їх установлення, налаштування та технічну підтримку. У межах виконання науково-дослідної роботи за темою «Метод і засіб захисту програмного забезпечення з апаратною прив’язкою до параметрів оперативної пам’яті та TPM-модуля» використовувалося програмне забезпечення, яке є безкоштовним або має відкриту ліцензію. Придбання комерційних продуктів не здійснювалося, тому фактичні витрати є мінімальними.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$В_{прг} = \sum_{i=1}^{k} (Ц_i^{прг} \cdot n_i \cdot K_i) \tag{4.11}$$

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

$C_{прг.і}$ – кількість одиниць ПЗ відповідного найменування, шт.;

K_i – коефіцієнт інсталяції та налаштування (приймемо $K_i = 1{,}05$);

k – кількість найменувань програмних засобів.

$B_{прг} = 300{,}00 \cdot 1 \cdot 1{,}05 = 315{,}00$ грн.

Отримані результати зведемо до таблиці:

Таблиця 4.9 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт.	Ціна за одиницю, грн	Вартість з урахуванням K_i, грн
Доступ до мережі Internet (високошвидкісний), грн/місяць	1	300,00	315,00
OpenJDK 17 (платформа Java SE)	1	0,00	0,00
Інтегроване середовище розробки IntelliJ IDEA Community Edition	1	0,00	0,00
Криптографічна бібліотека BouncyCastle	1	0,00	0,00
TPM/JAVA API (tpm2-tools bindings)	1	0,00	0,00
Система віртуалізації VirtualBox	1	0,00	0,00
Допоміжні інструменти (Wireshark, Notepad++, 7-Zip)	1	0,00	0,00
Всього:			315,00

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 та приміщень, грн;

$t_{вик}$ – термін використання об’єкта під час НДР, міс.;

$T_в$ – строк корисного використання, років.

$$A_{\text{ПК}} = \frac{38900{,}00 \cdot 1}{4 \cdot 12} = 810{,}42 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 4.10 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання, міс.	Амортизаційні відрахування, грн
Робоча станція розробника (Intel i7 / 32 GB RAM / 1 TB NVMe / RTX 4060)	38900,00	4	1	810,42
Ноутбук інженера з безпеки (HP ProBook 450 G9)	29500,00	4	1	614,58
Персональний ПК аналітика (Ryzen 5 / 16 GB RAM / 512 GB SSD)	22500,00	5	1	375,00
Принтер/сканер HP LaserJet MFP 135w	8200,00	4	1	170,83
Комутатор TP-Link TL-SG108E (8-port)	1890,00	4	1	39,38
Маршрутизатор ASUS RT-AX55	3790,00	4	1	78,96
USB-ключі для експериментів (набір з 4 шт.)	1200,00	3	1	33,33
Програмне забезпечення Windows 11 Pro	7800,00	3	1	216,67
Пакет Microsoft Office Home & Business	6200,00	3	1	172,22
Приміщення лабораторії (умовна оціночна вартість)	420000,00	25	1	1400,00
Оргтехніка (клавіатури, миші, периферія)	6500,00	5	1	108,33
Всього:				4019,72

4.3.8 Паливо та енергія для науково-виробничих цілей

До цієї статті належать витрати на електроенергію, що використовується обладнанням під час проведення НДР. Витрати на електроенергію ($В_е$) визначаємо за формулою (4.13):

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

де:

$N_{вст}$– встановлена потужність обладнання, кВт;

t_i– тривалість роботи обладнання на етапі досліджень, год;

C_e– вартість 1 кВт·год електроенергії, грн (12,56 грн);

$K_{вп}$– коефіцієнт використання потужності (0,92 … 0,97);

η_i– коефіцієнт корисної дії обладнання (0,95 … 0,98)

$$B_{e(ПК)} = 0{,}32 \cdot 180 \cdot 12{,}56 \cdot 0{,}95/0{,}97 = 708{,}34 \text{ грн.};$$

$K_{вni}$ – 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.

Таблиця 4.11 – Витрати на електроенергію

Найменування обладнання	Потужність, кВт	Тривалість роботи, год	Сума, грн
Робоча станція розробника (Intel i7 / 32 GB / RTX 4060)	0,32	180	708,34
Ноутбук інженера з безпеки HP ProBook 450 G9	0,09	180	209,33
ПК аналітика систем захисту (Ryzen 5)	0,12	180	279,98
Принтер/сканер HP LaserJet MFP 135w	0,25	5	15,70
Маршрутизатор ASUS RT-AX55	0,02	180	46,11
Комутатор TP-Link TL-SG108E	0,015	180	34,58
USB-обладнання та периферія (сумарно)	0,03	180	68,97
Оргтехніка (офісне обладнання)	0,10	12	15,07
Всього:			1378,08

4.3.9 Службові відрядження

До статті «Службові відрядження» належать витрати на відрядження працівників, які залучені до виконання науково-дослідної роботи за темою

«Метод та засіб захисту програмного засобу методом прив’язки до оперативної пам’яті».

До таких витрат можуть належати:

- виїзди для збору інформації щодо апаратних платформ, механізмів керування пам’яттю та специфікацій ОС;
- участь у наукових семінарах, конференціях, присвячених дослідженням у сфері захисту пам’яті, аналізу шкідливих впливів та реверс-інжинірингу;
- відрядження для проведення незалежного тестування засобу на сторонніх апаратно-програмних стендах;
- консультації з експертами з низькорівневих механізмів Windows та безпеки оперативної пам’яті.

У ході виконання даної НДР потреби у відрядженнях не виникало, оскільки:

- експериментальне середовище було повністю розгорнуте в лабораторії;
- доступ до необхідних технічних матеріалів та консультацій було забезпечено дистанційно;
- тестування засобу прив’язки до оперативної пам’яті виконувалося локально на наявному обладнанні.

Витрати за цією статтею приймаємо рівними нулю.

Витрати на службові відрядження (Всв) визначаються за формулою (4.14):

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

де

$З_о$– основна заробітна плата дослідників;

$З_р$– основна заробітна плата робітників;

$Н_{св}$– норма витрат за статтею «Службові відрядження» (20…25%), приймемо 0%, оскільки відряджень не здійснювалося.

Оскільки відряджень не здійснювалося, приймаємо:

$$H_{св} = 0\%.$$

Тоді:

$$B_{св} = (58354{,}55 + 5636{,}31) \cdot 0 = 0 \; грн.$$

У рамках виконання НДР службові відрядження не проводилися, тому витрати за статтею «Службові відрядження» дорівнюють нулю.

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

До статті «Витрати на роботи, які виконують сторонні підприємства, установи і організації» належать витрати, пов’язані з виконанням окремих етапів науково-дослідної роботи зовнішніми спеціалізованими структурами.

У рамках дослідження за темою «Метод та засіб захисту програмного засобу методом прив'язки до оперативної пам'яті» сторонні організації можуть бути залучені для таких робіт:

- проведення незалежного аудиту стійкості методу прив’язки до оперативної пам’яті;
- експертний аналіз алгоритмів виявлення модифікацій пам’яті;
- консультації з фахівцями у сфері низькорівневого доступу до пам’яті та реверс-інжинірингу;
- незалежне тестування протидії атакам (memory injection, memory cloning);
- використання лабораторного стенду стороннього закладу для експериментальної перевірки.

Витрати на роботи сторонніх організацій (Всп) визначаються за формулою (4.15):

$$\text{В}_{\text{сп}} = (\text{З}_{\text{о}} + \text{З}_{\text{р}}) \cdot \frac{\text{Н}_{\text{сп}}}{100}, \qquad (4.15)$$

де:

$\text{З}_{\text{о}}$– основна заробітна плата дослідників;

$\text{З}_{\text{р}}$– основна заробітна плата робітників;

$Н_{сп}$– норма витрат за статтею «Витрати на роботи, які виконують сторонні організації». Для теми приймемо Нсп = 30%.

$$В_{сп} = (58354,55 + 5636,31) \cdot 0,30 =$$

$$= 63990,86 \cdot 0,30 = 19197,26 \text{ грн.}$$

Витрати на роботи, що виконуються сторонніми організаціями та пов’язані з дослідженням методу прив’язки програмного засобу до оперативної пам’яті, складають: 19197,26 грн..

4.3.11 Інші витрати

До статті «Інші витрати» належать витрати, які не були враховані в попередніх статтях, але можуть бути безпосередньо віднесені на собівартість виконання науково-дослідної роботи за темою «Метод та засіб захисту програмного засобу методом прив’язки до оперативної пам’яті».

У цю групу можуть входити:

- витрати на допоміжні матеріали та дрібні інструменти, не включені до статті «Сировина та матеріали»;
- витрати, пов’язані з обслуговуванням робочих місць під час проведення експериментів;
- канцелярські та організаційні витрати;
- витрати на хмарні сервіси або локальні ресурси корпоративних мереж (за потреби);
- витрати на непередбачувані технологічні потреби.

Беручи до уваги, що дослідницький процес включав роботу з низькорівневими механізмами обробки оперативної пам’яті, багатократні цикли тестування, налаштування експериментального середовища та допоміжні операції, інші витрати доцільно оцінювати на рівні 50% від суми основної заробітної плати дослідників і робітників.

Витрати за статтею «Інші витрати» розраховуємо за формулою (4.16):

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

де:

$З_о$– основна заробітна плата дослідників;

$З_р$– основна заробітна плата робітників;

$Н_{ів}$– норма нарахування за статтею «Інші витрати» (50…100%).

$$I_в = (58354{,}55 + 5636{,}31) \cdot 0{,}50 = 31995{,}43 \text{ грн.}$$

4.3.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать витрати, які забезпечують загальні умови виконання науково-дослідної роботи і не можуть бути віднесені безпосередньо до окремого етапу. Для дослідження за темою «Метод та засіб захисту програмного забезпечення методом прив'язки до оперативної пам'яті» до накладних витрат можуть входити:

- витрати на організаційне та адміністративне управління проектом;
- витрати на забезпечення функціонування інфраструктури, мережевих ресурсів і серверного обладнання;
- оплата банківських та сервісних послуг (зокрема хостинг, домен, доступ до наукових баз даних);
- витрати на підготовку та підвищення кваліфікації учасників дослідження;
- витрати на організацію технічних консультацій та внутрішніх нарад;
- витрати на інформаційне забезпечення, рекламу або просування результатів.

Згідно методичних рекомендацій, накладні (загальновиробничі) витрати визначаються у розмірі 100…150% від суми основної заробітної плати дослідників і робітників.

Для даної роботи приймаємо:

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

де

$З_о = 58354{,}55$грн – основна заробітна плата дослідників;

$З_р = 5636{,}31$грн – основна заробітна плата робітників;

$Н_{нзв} = 100\%$.

$$B_{нзв} = (58354{,}55 + 5636{,}31) \cdot 1{,}0 = 63990{,}85 \text{ грн.}$$

Загальні витрати на проведення НДР визначаються сумою всіх обчислених статей витрат:

$B_{заг}$=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, що враховує стадію виконання НДР, приймемо η=0,9.

Тоді:

$$ЗВ = \frac{230978{,}24}{0{,}9} = 256642{,}49 \text{ грн..}$$

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

Метою цього підрозділу є визначення економічної доцільності впровадження та комерціалізації науково-технічної розробки «Метод та засіб захисту програмного забезпечення методом прив’язки до оперативної пам’яті» потенційним інвестором. Розрахунок виконується відповідно до (Козловського, Леська, Кавецького (ВНТУ, 2021).

Комерційна ефективність визначається на основі таких показників:

- прогнозованих витрат на впровадження та промислову експлуатацію розробки;
- потенційних доходів від продажу або ліцензування засобу захисту;
- величини чистого прибутку;
- строку окупності інвестицій;
- рівня рентабельності.

4.4.1 Визначення річного економічного ефекту

Для оцінювання економічного ефекту в методичці застосовується формула:

$$Е = Д - В, \quad (4.20)$$

де

Д – річний дохід від реалізації продукту, грн;

В – сума річних витрат, грн.

Прийняті дані (обґрунтовані для ПЗ із захистом методом RAM-binding)

Оскільки розробка належить до засобів кібербезпеки, орієнтованих на:

- розробників програмних продуктів,
- ІТ-компанії та фірми, що захищають ПЗ від реверс-інжинірингу,
- системних інтеграторів, то пропонується економічно обґрунтована модель продажу ліцензій.

Припустимо, що:

- вартість однієї комерційної ліцензії: 1200 грн;
- кількість ліцензій, що може бути реалізована протягом першого року: 600 штук.

Тоді:

$$Д = 1200 \cdot 600 = 720000 \text{ грн.}$$

4.4.2 Річні витрати на впровадження

Згідно методичних вказівок, річні витрати на впровадлення включають:

- витрати на завершення НДР (попередньо розраховано – 256642,49 грн),
- витрати на технічну підтримку (20–25% від вартості НДР),
- витрати на адаптацію, просування, документацію.

Приймемо норму обслуговування:

$$Н_{тп} = 25\%.$$

Тоді:

$$В_{тп} = 256642{,}49 \cdot 0{,}25 = 64160{,}62 \text{ грн.}$$

Загальна сума річних витрат:

$$В = 256642{,}49 + 64160{,}62 = 320803{,}11 \text{ грн.}$$

4.4.3 Розрахунок річного економічного ефекту

$$E = 720000 - 320803{,}11 = 399196{,}89 \text{ грн.}$$

Річний економічний ефект позитивний і суттєвий.

4.4.4 Розрахунок чистого прибутку

Згідно методички, чистий прибуток визначається як:

$$П_ч = Е \cdot (1 - k), \qquad (4.21)$$

де

$k = 0{,}18$– ставка податку на прибуток.

Тоді:

$$П_ч = 399196{,}89 \cdot (1 - 0{,}18) = 327332{,}45 \text{ грн.}$$

4.4.5 Розрахунок строку окупності інвестицій

Строк окупності визначається за формулою:

$$T_{ок} = \frac{ЗВ}{П_ч}, \qquad (4.22)$$

де

$ЗВ = 256642{,}49$грн – загальні витрати на виконання НДР.

$$T_{ок} = \frac{256642{,}49}{327332{,}45} = 0{,}78 \text{ року}$$

4.4.6 Рівень рентабельності

Рентабельність:

$$R = \frac{П_ч}{В} \cdot 100\%. \qquad (4.23)$$

$$R = \frac{327332{,}45}{320803{,}11} \cdot 100\% = 102{,}0\%.$$

Таким чином, інвестиції окупляться за 9–10 місяців, що є дуже високим показником.

Проведені розрахунки показують, що науково-технічна розробка «Метод та засіб захисту програмного забезпечення методом прив’язки до оперативної пам’яті» є економічно привабливою для потенційного інвестора.

Отримані результати:

- річний економічний ефект: 399196,89 грн;
- чистий прибуток: 327332,45 грн;

- строк окупності: 0,78 року (менше 1 року);
- рівень рентабельності: ≈102%.

Це свідчить, що розробка має високий комерційний потенціал, а її впровадження забезпечить значний економічний ефект при мінімальних ризиках.

4.5 Висновки до розділу

Згідно з проведеним комерційним та технологічним аудитом встановлено, що рівень комерційного потенціалу розробки за темою «Метод та засіб захисту програмного забезпечення методом прив’язки до оперативної пам’яті» становить 40,3, що відповідно до шкали оцінювання свідчить про високий рівень науково-технічної перспективності та комерційної значущості розробки. Це підтверджує доцільність подальшої роботи над її впровадженням та можливу зацікавленість потенційних інвесторів.

За результатами технічного аналізу встановлено, що узагальнений коефіцієнт якості нового рішення перевищує технічний рівень існуючих аналогів приблизно в 1.25. Переваги забезпечені використанням унікальної комбінації RAM-профілювання, апаратної TPM-валідації, криптографічного контейнера та багаторівневої системи обфускації. Такий підхід формує суттєво вищу стійкість до атак клонування, підміни апаратної конфігурації та модифікації контейнера, що не реалізується у традиційних засобах програмного захисту.

Економічні розрахунки показали, що розробка є економічно доцільною та має високий інвестиційний потенціал, оскільки строк окупності становить 0,69 року, що значно менше за нормативне значення. Результати підтверджують, що розроблений метод і програмний засіб є технологічно інноваційними та перспективними для подальшого впровадження.

ВИСНОВКИ

У магістерській кваліфікаційній роботі вирішено актуальну задачу підвищення рівня захисту програмного забезпечення від несанкціонованого запуску, модифікації та клонування шляхом використання апаратно унікальних характеристик комп'ютерної системи. Запропоновано метод прив'язки виконуваного коду до параметрів оперативної пам'яті (RAM) та модуля довіреної платформи (TPM), що забезпечує формування унікального та невідтворюваного профілю середовища виконання.

1. Проведений аналіз сучасних підходів засвідчив, що традиційні методи захисту (серійні ключі, обфускація, прив'язка до MAC-адреси чи HDD) не забезпечують належної стійкості до атак з використанням віртуалізації та емуляції обладнання. Обґрунтовано, що використання параметрів RAM, які не можуть бути коректно відтворені програмними засобами, створює надійну основу для побудови апаратно орієнтованої автентифікації програмного забезпечення.
2. Розроблено метод формування апаратно-криптографічного профілю середовища виконання (DeviceHash), що поєднує SPD-дані, таймінги та інші фізичні параметри оперативної пам'яті з атестованими TPM-артефактами. Показано, що гешування SHA-256 забезпечує високу ентропію ідентифікатора і чітку залежність від конфігурації системи, що унеможливлює клонування або перенесення захищеної програми на інший комп'ютер.
3. Розроблено алгоритм захищеного виконання програмного коду, у якому контейнер із зашифрованими класами розшифровується лише за умови відповідності апаратного профілю. Для шифрування застосовано алгоритм Serpent, який забезпечує високу стійкість до криптоаналізу завдяки 32 раундам обробки даних та стійкому ключовому розкладу. Ключ шифрування генерується динамічно з DeviceHash, що виключає можливість його відновлення на іншому обладнанні.

4. Створено програмний засіб, що складається з модулів SecurityMKR та Launcher. Реалізовано механізм SecureLoader для завантаження класів безпосередньо в оперативну пам’ять без створення тимчасових файлів. Цей підхід мінімізує ризики статичного аналізу, перехоплення та модифікації програмного коду інструментами файлового моніторингу, що є важливим у контексті сучасних атак на ПЗ.
5. Експериментальні дослідження продемонстрували високу практичну ефективність запропонованого рішення. Система коректно реагувала на заміну модулів RAM, перенесення на віртуальні машини та спроби модифікації зашифрованого контейнера. Точність автентифікації апаратного середовища становить близько 98%. Узагальнений коефіцієнт якості розробки дорівнює 2,06, що суттєво перевищує показники існуючих аналогів захисту.
6. Економічні розрахунки свідчать про високу доцільність впровадження розробки у практичні системи. Строк окупності становить 0,78 року, рентабельність – понад 100%, а прогнозований річний економічний ефект складає близько 399 тис. грн. Це підтверджує інвестиційну привабливість та комерційний потенціал розробленого методу і програмного засобу.

У підсумку доведено, що запропонований метод автентифікації середовища виконання на основі апаратних характеристик RAM та TPM забезпечує високий рівень захисту програмного забезпечення від клонування та несанкціонованого використання. Отримані результати мають як теоретичну, так і практичну цінність та можуть бути основою для подальших досліджень у напрямі створення багатофакторних систем контролю середовища виконання, розробки апаратно-криптографічних протоколів та вдосконалення систем протидії реверс-інжинірингу та модифікації програмного забезпечення.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. TPM 2.0: A Brief Introduction [Електронний ресурс] // Trusted Computing Group. – 2020. URL: https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-2.0-Library-Part-0-Version-184_pub.pdf (дата звернення: 05.10.2025).
2. L. Kuperstein, V. Lutsyshyn // Кібербезпека: освіта, наука, техніка. Інформаційна технологія моніторингу безпеки данихпрограмного забезпечення – 2019. URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/562/440 (дата звернення: 05.10.2025).
3. Saleh K. Hardware and Software Methods for Secure Obfuscation and Deobfuscation: An In-Depth Analysis [Електронний ресурс] / K. Saleh // Computers. – 2025. – Vol. 14, Iss. 7. – 251. URL: https://www.mdpi.com/2073-431X/14/7/251 (дата звернення: 05.10.2025).
4. Secure PUF-Based Authentication Systems [Електронний ресурс] // PubMed Central (PMC). – 2024. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11487452/ (дата звернення: 07.10.2025).
5. Zero Trust Architecture [Електронний ресурс] : NIST Special Publication 800-207 / S. Rose [et al.]. – Gaithersburg : National Institute of Standards and Technology, 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf (дата звернення: 11.10.2025).
6. Evaluation Methodologies in Software Protection Research [Електронний ресурс] / B. De Sutter [et al.] // ACM Computing Surveys. – 2024. URL: https://dl.acm.org/doi/pdf/10.1145/3702314 (дата звернення: 11.10.2025).
7. Validating the Integrity of Computing Devices [Електронний ресурс]. – Gaithersburg : National Institute of Standards and Technology, 2022. – (NIST Special Publication 1800-34). URL:

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1800-34.pdf (дата звернення: 11.10.2025).

8. A comprehensive review of IoT device fingerprinting: Insights into techniques, trends, challenges, and future directions [Електронний ресурс] // ScienceDirect. – 2025. URL: https://www.sciencedirect.com/science/article/pii/S2542660525002719 (дата звернення: 13.10.2025).
9. Software Piracy Protection Using Serial Key Generation and Verification for an Individual User [Електронний ресурс] // ResearchGate. – 2023. URL: https://www.researchgate.net/publication/376262986_Software_Piracy_Protection_Using_Serial_Key_Generation_and_Verification_for_an_Individual_User (дата звернення: 13.10.2025).
10. Cryptographic Hash Functions [Електронний ресурс] // GeeksforGeeks. – 2025. URL: https://www.geeksforgeeks.org/competitive-programming/cryptography-hash-functions/ (дата звернення: 21.10.2025).
11. Бойко О. В. Порівняльний аналіз криптографічних алгоритмів: RSA, AES та їх гібридне використання / О. В. Бойко, В. С. Яремчук // Вісник Національного технічного університету України «КПІ». Серія «Інформатика, управління та обчислювальна техніка». – 2020. – № 6. – С. 115–123. (дата звернення: 21.10.2025).
12. USB Tokens [Електронний ресурс] // Certinal Glossary. – URL: https://www.certinal.com/glossary/what-are-usb-tokens (дата звернення: 21.10.2025).
13. Sánchez Sánchez P. M. та ін. A methodology to identify identical single board computers through hardware behavioral fingerprinting // Journal of Network and Computer Applications. – 2023. – Vol. 214. (дата звернення: 23.10.2025).
14. Trusted Platform Module Technology Overview [Електронний ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/security/hardware-security/tpm/trusted-platform-module-overview (дата звернення: 23.10.2025).

15. Secure Boot [Електронний ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-secure-boot (дата звернення: 23.10.2025).
16. Intel® Software Guard Extensions (Intel® SGX) Overview [Електронний ресурс] // Intel. – URL: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html (дата звернення: 25.10.2025).
17. JESD400-5B. DDR5 Serial Presence Detect (SPD) Contents [Електронний ресурс]. – Arlington : JEDEC Solid State Technology Association, 2023. – URL: https://standards.globalspec.com/std/14635223/jesd400-5b (дата звернення: 28.10.2025).
18. Xiong W. та ін. Dynamic Physically Unclonable Functions // Proc. Great Lakes Symposium on VLSI (GLSVLSI'19). – New York : ACM, 2019. – P. 1–4. (дата звернення: 29.10.2025).
19. Foti J. Advanced Encryption Standard (AES) [Електронний ресурс] : FIPS PUB 197-upd1. – Gaithersburg : National Institute of Standards and Technology, 2022. – URL: https://doi.org/10.6028/nist.fips.197-upd1.ipd (дата звернення: 01.11.2025).
20. Chen L. S. та ін. Recommendation for Block Cipher Modes of Operation: Methods for Confidentiality and Authenticated Encryption [Електронний ресурс] : NIST Special Publication 800-38G. – Gaithersburg : National Institute of Standards and Technology, 2021. – URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38G.pdf (дата звернення: 01.11.2025).
21. Богданов В. В., Мельников В. О. Геш-функції в системах інформаційної безпеки: основи та застосування. – Київ : Центр навчальної літератури, 2021. – 280 с. (дата звернення: 04.11.2025).
22. Кириленко М. О. Використання криптографії на еліптичних кривих у механізмах безпечного завантаження (Secure Boot) // Кібербезпека та сучасні інформаційні технології : матеріали міжнар. наук.-практ. конф. –

Харків, 2022. – С. 40–44. (дата звернення: 04.11.2025).

23. Cuzzocrea A., Iacoviello M. M. Classification and comparison of software protection techniques [Електронний ресурс] // Journal of Systems and Software. – 2019. – Vol. 157. – Art. 110410. – URL: https://doi.org/10.1016/j.jss.2019.110410 (дата звернення: 04.11.2025).
24. 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: 06.11.2025).
25. Кавецький В. В., Козловський В. О., Причепа І. В. Економічне обґрунтування інноваційних рішень: практикум. Вінниця : ВНТУ, 2016. 113 с. (дата звернення: 06.11.2025).

ДОДАТКИ

Додаток А

Технічне завдання

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

ЗАТВЕРДЖУЮ
В.о. зав. каф. ЗІ, д. т. н., професор
_____В. А. Лужецький

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання магістерської кваліфікаційної роботи

на тему: «МЕТОД І ЗАСІБ ЗАХИСТУ ПРОГРАМНОГО ЗАСОБУ ЗА РАХУНОК ПРИВ'ЯЗКИ ДО ОПЕРАТИВНОЇ ПАМ'ЯТІ»
08-53.МКР.023.03.000 ТЗ

Керівник магістерської кваліфікаційної
роботи доц. каф. ЗІ, к. т. н., доц.
_____ Володимир ГАРНАГА

м. Вінниця – 2025 рік

1 Підстави для проведення робіт

Робота проводиться на підставі наказу ректора ВНТУ від 24 вересня 2025 року № 313 про затвердження індивідуального завдання на магістерську кваліфікаційну роботу.

Дата початку роботи 09.09.25 р.

Дата закінчення роботи 19.12.25 р.

2 Мета та призначення МКР

Мета – підвищення рівня захисту програмного забезпечення за рахунок створення методу та засобу апаратно-програмної прив'язки виконуваного коду до унікальних характеристик оперативної пам'яті (RAM) та модуля довіреної платформи (TPM).

Об'єктом процес апаратно-програмної автентифікації середовища виконання програмного забезпечення.

Предметом методи та засоби захисту програмного забезпечення на основі апаратних характеристик пам'яті та TPM.

Актуальність теми Зі зростанням складності атак на програмне забезпечення та широким використанням віртуальних середовищ, традиційні механізми захисту (ліцензійні ключі, серійні номери, обфускація) втрачають ефективність. Сучасні засоби клонування та емуляції дозволяють відтворити більшість стандартних параметрів системи. Натомість оперативна пам'ять містить набір фізичних параметрів (таймінги, SPD-дані, структуру модулів), що не можуть бути точно підроблені чи емітовані. Поєднання цих властивостей з криптографічною атестацією TPM дає можливість побудови високонадійного методу автентифікації середовища виконання, що є критично важливим для захисту цінного, комерційного та індустріального програмного забезпечення.

3 Вихідні дані для проведення МКР

МКР проводиться вперше і вхідними даними для проведення МКР є:

3.1 Asaduzzaman M. et al. Trusted Hardware Identifiers for Software Licensing // *IEEE Security & Privacy*. 2020. Vol. 18, no. 3.

3.2 Tehranipoor M. M., Guin U. Entropy Sources in DRAM-Based PUFs: Analysis and Improvements // *IEEE Transactions on Information Forensics and Security*. 2020. Vol. 15. P. 1621–1634.

3.3 Xu J., Yao Y., Wyglinski A. M. A Survey on Software Obfuscation: Attacks, Defenses, and Evaluations // *IEEE Access*. 2021. Vol. 9. P. 156356–156385.

4 Виконавці МКР

Студент групи 1БС-24м Соколюк Олександр Миколайович

5 Вимоги до виконання МКР

Для досягнення поставленої мети необхідно вирішити такі задачі:

– виконати огляд та аналіз сучасних методів захисту програмного забезпечення;

– виявити обмеження традиційних систем ліцензування та методів

прив’язки до пристроїв;
- розробити метод формування апаратно-криптографічного профілю RAM;
- сформувати математичну модель автентифікації середовища виконання;
- розробити алгоритми гешування та шифрування програмного контейнера;
- розробити програмний засіб у складі модулів SecurityMKR та Launcher;
- реалізувати механізм SecureLoader для виконання коду без створення тимчасових файлів;
- провести експериментальні дослідження (фізичне заміщення RAM, запуск на ВМ, модифікація контейнера);
- виконати техніко-економічне обґрунтування доцільності розробки;
- підготувати висновки щодо ефективності запропонованого методу.

6 Вимоги до супровідної документації

Графічна і текстова документація повинна відповідати діючим стандартам України – ДСТУ 3008:2015.

7 Етапи МКР

Робота з теми виконується за такими етапами:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	01.09.2024 – 04.09.2024	
2	Аналіз інформаційних джерел за напрямком магістерської кваліфікаційної роботи	05.09.2024 – 15.09.2024	
3	Науково-технічне обґрунтування	16.09.2024 – 22.09.2024	
4	Розробка технічного проєкту системи захисту	23.09.2024 – 04.10.2024	
5	Розробка математичної моделі та методу апаратної прив’язки	05.10.2024 – 08.10.2024	
6	Розробка криптографічних алгоритмів і механізму SecureLoader	09.10.2024 – 16.10.2024	
7	Експериментальні дослідження та аналіз результатів	17.10.2024 – 14.11.2024	
8	Розробка розділу економічного обґрунтування доцільності розробки	18.11.2024 – 21.11.2024	
9	Оформлення пояснювальної записки	25.11.2024 – 29.11.2024	
10	Перевірка магістерської роботи на наявність текстових запозичень	30.11.2024 – 02.12.2024	
11	Попередній захист та доопрацювання МКР	03.12.2024 – 14.12.2024	
12	Представлення МКР до захисту, рецензування	15.12.2024 – 20.12.2024	
13	Захист МКР	21.12.2024 – 23.12.2024	

8 Очікувані результати та порядок реалізації МКР

Передбачається створення методу апаратно-криптографічного захисту ПЗ на основі апаратних характеристик RAM та TPM, а також

програмного засобу, який може бути використаний у практичних системах захисту інформації та у навчальному процесі.

9 Матеріали які подаються після закінчення МКР

По завершенню роботи подається пояснювальна записка та ілюстративна частина.

10 Порядок приймання МКР та її етапів

Апробація на науково-технічних конференціях та семінарах. Результати ро- боти будуть розглядатися на засіданні ДЕК із захисту магістерських квалі- фікаційних робіт.

Попередній захист та доопрацювання МКР – 5-7 листопада 2025 р.
Представлення МКР до захисту – 15-18 грудня 2025 р.
Захист МКР – 19-21 грудня 2025 р.

11 Вимоги до розроблення документації

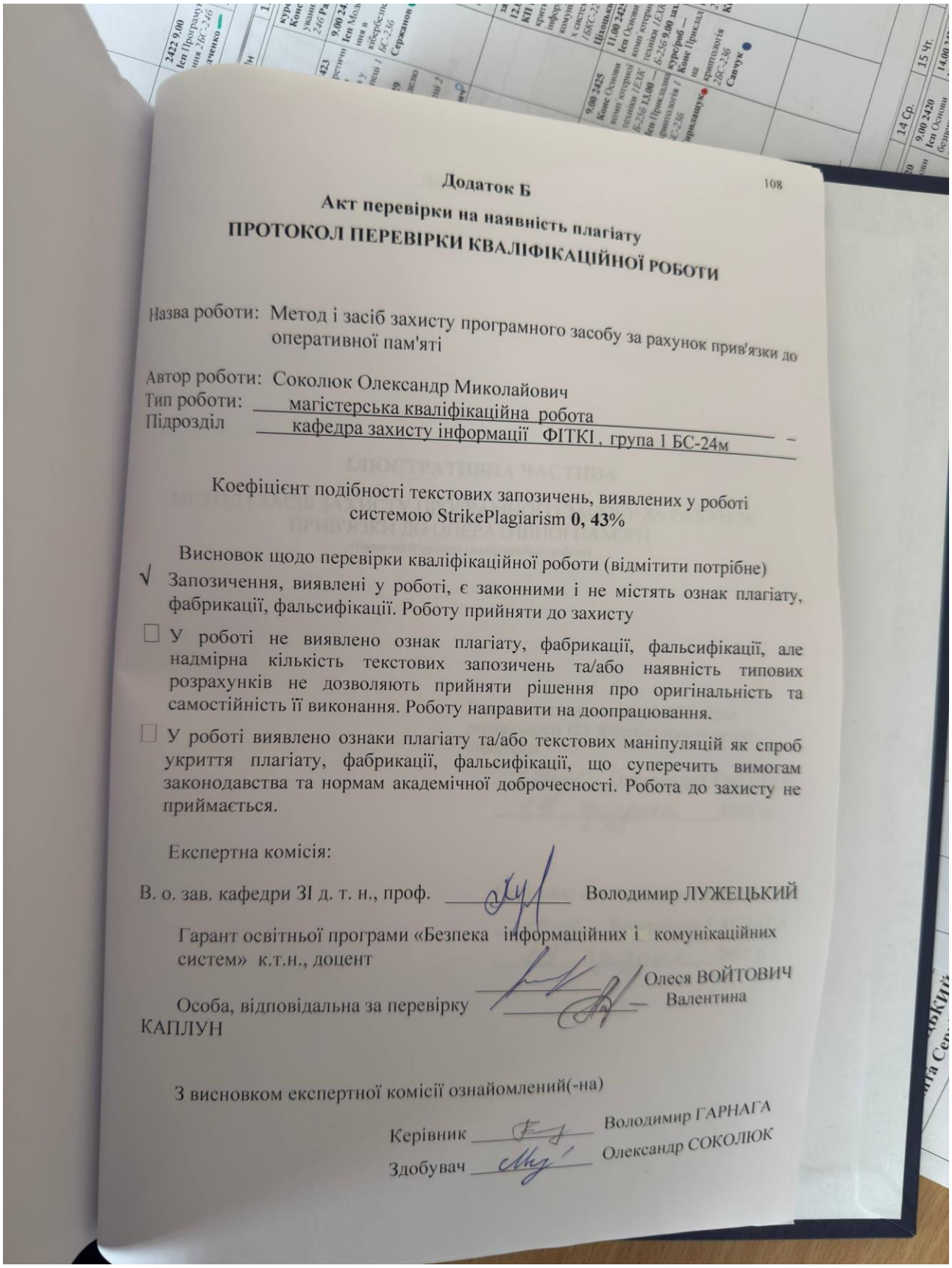
Документація буде виконуватись за допомогою комп'ютерного набору у відповідності вимог ДСТУ 3008:2015 «Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання».

12 Вимоги щодо технічного захисту інформації з обмеженим доступом

У зв'язку з тим, що дана робота не містить інформації, що потребує захисту у відповідності до законів України, заходи з її технічного захисту не перед- бачаються.

Розробив студент групи 1БС-24м _____Олександр СОКОЛЮК

Додаток Б



Додаток Б 108

Акт перевірки на наявність плагіату

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Метод і засіб захисту програмного засобу за рахунок прив'язки до оперативної пам'яті

Автор роботи: Соколюк Олександр Миколайович
Тип роботи: магістерська кваліфікаційна робота
Підрозділ кафедра захисту інформації ФІТКІ, група 1 БС-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism **0, 43%**

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 і комунікаційних систем» к.т.н., доцент _____ Олеся ВОЙТОВИЧ

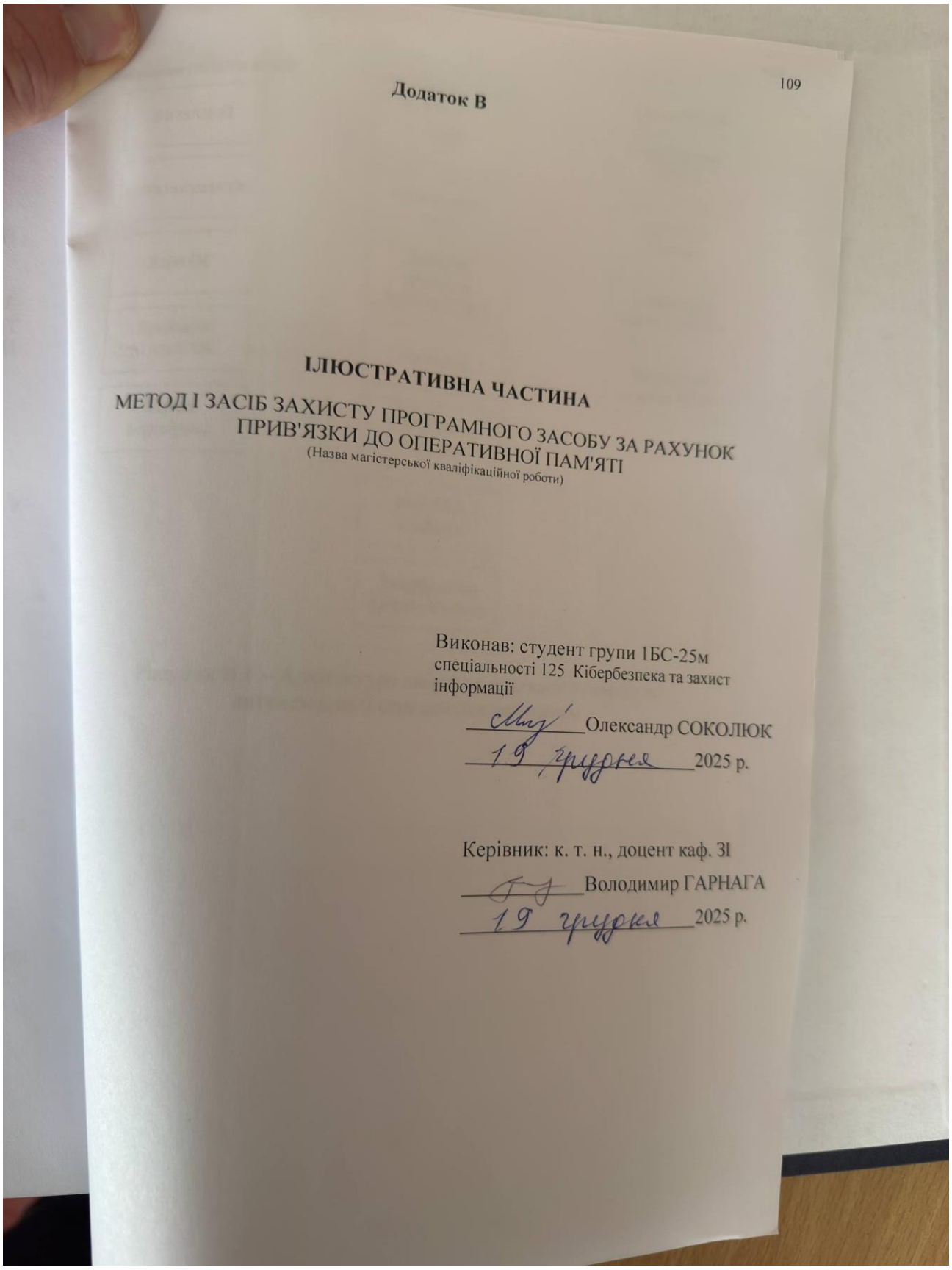
Особа, відповідальна за перевірку _____ Валентина КАПЛУН

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Володимир ГАРНАГА

Здобувач _____ Олександр СОКОЛЮК

Додаток В



Додаток В 109

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОД І ЗАСІБ ЗАХИСТУ ПРОГРАМНОГО ЗАСОБУ ЗА РАХУНОК ПРИВ'ЯЗКИ ДО ОПЕРАТИВНОЇ ПАМ'ЯТІ

(Назва магістерської кваліфікаційної роботи)

Виконав: студент групи 1БС-25м
спеціальності 125 Кібербезпека та захист інформації

_____ Олександр СОКОЛЮК
19 грудня 2025 р.

Керівник: к. т. н., доцент каф. ЗІ

_____ Володимир ГАРНАГА
19 грудня 2025 р.

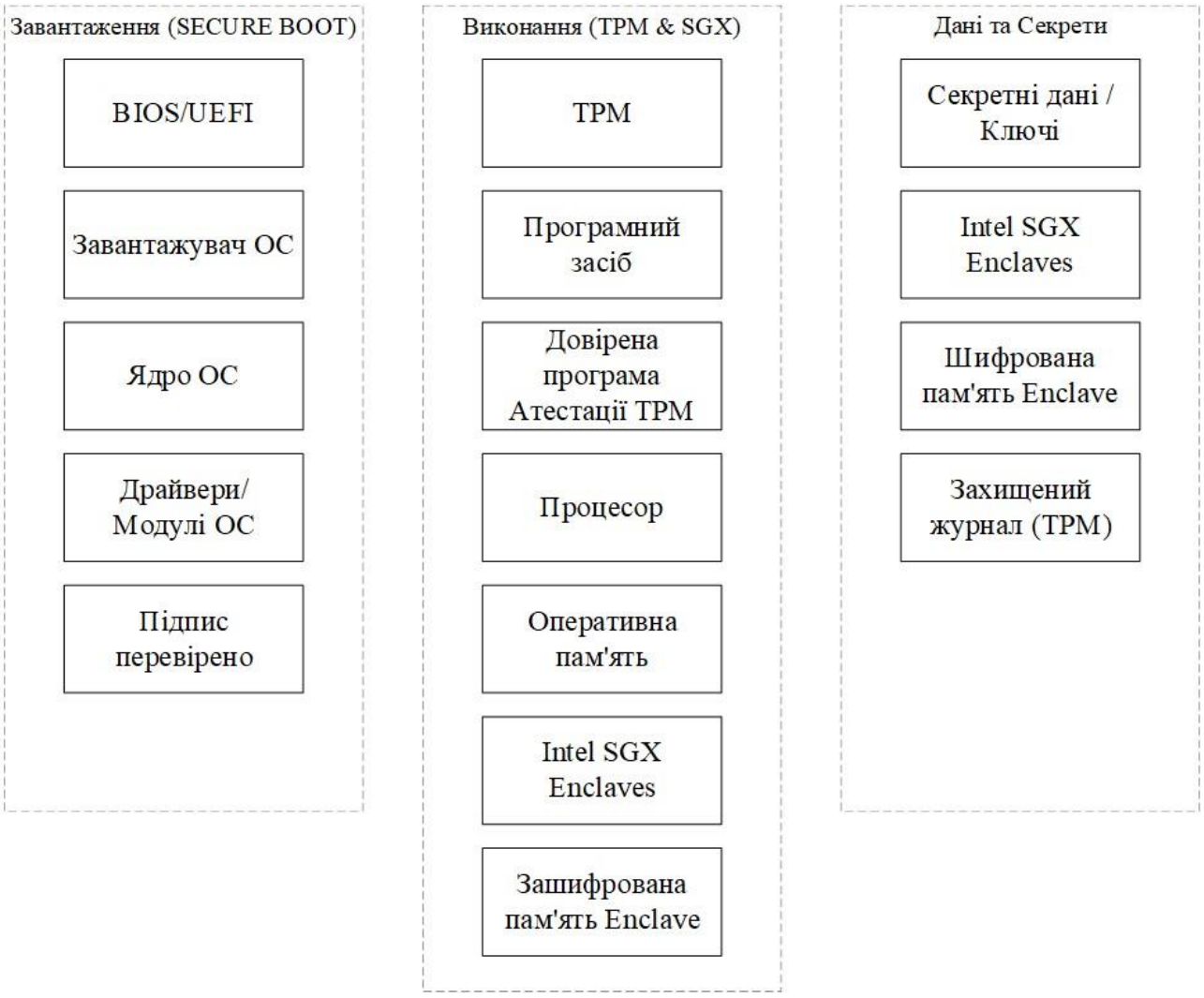


Рисунок В.1 – Архітектура апаратно-залежного контролю автентичності середовища виконання

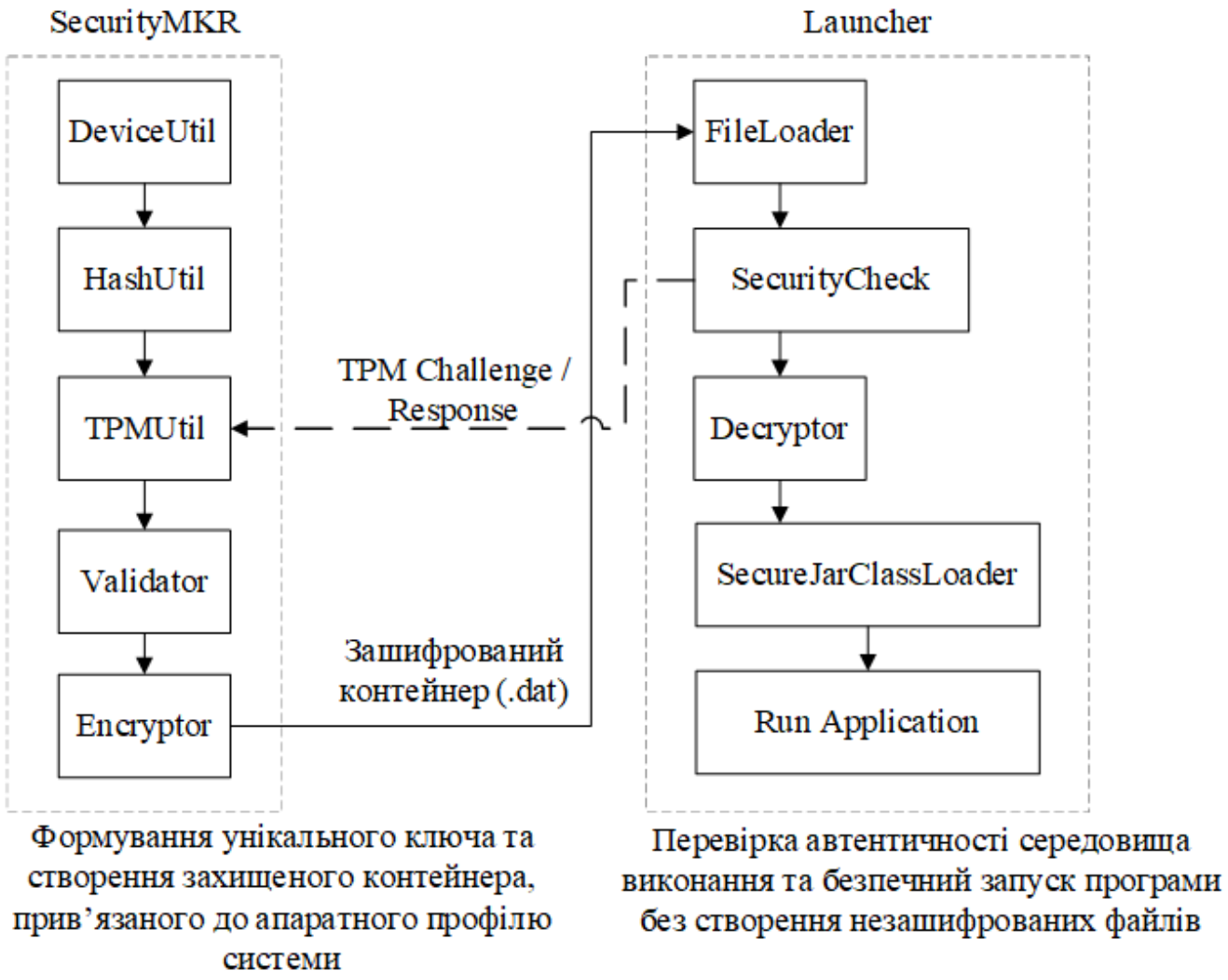


Рисунок В.2 – Загальна архітектура системи захисту програмного забезпечення

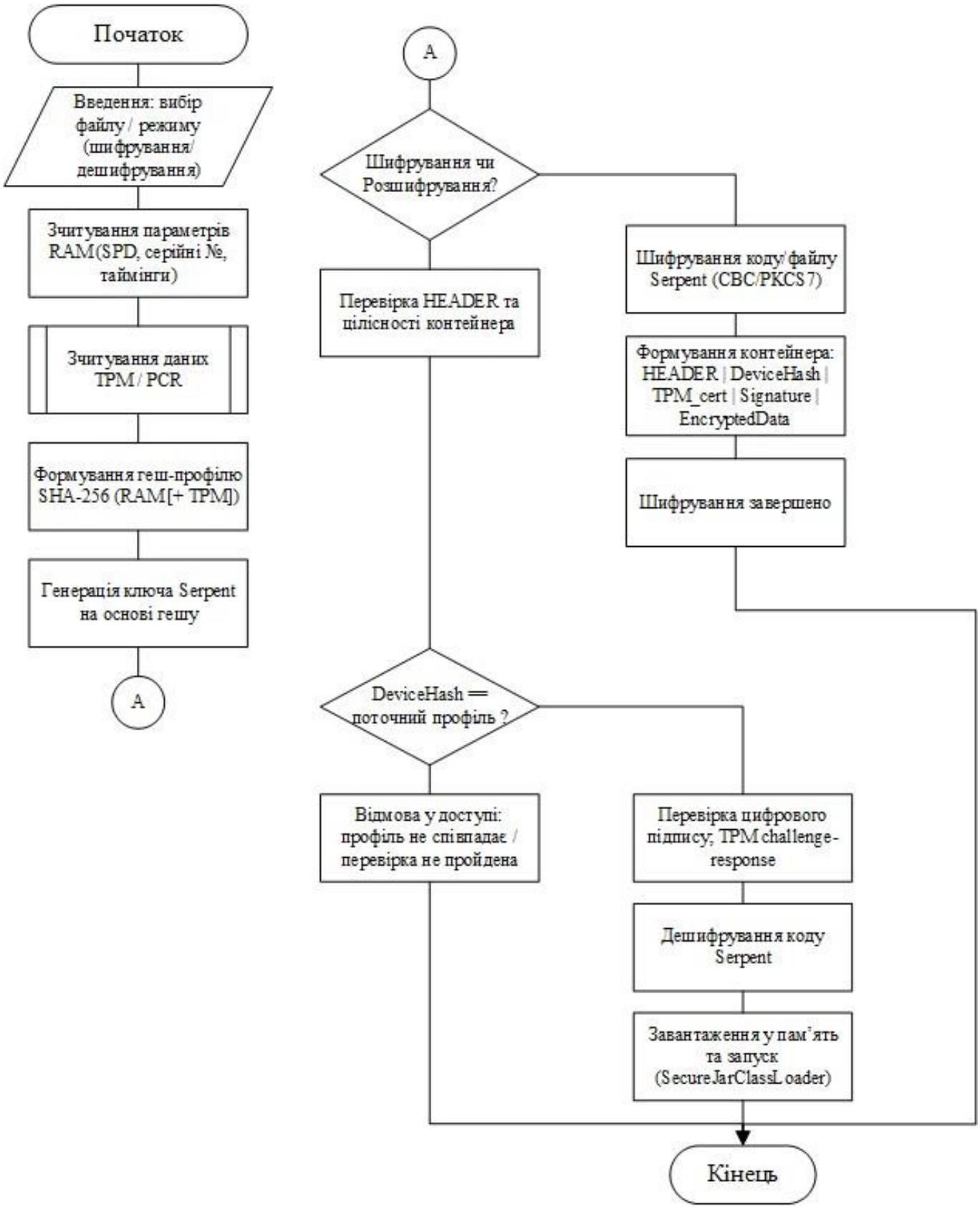


Рисунок В.3 – Загальний алгоритм методу прив'язки програмного забезпечення до оперативної пам'яті