Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Вдосконалення методу автентифікації у системах контролю версій вихідного коду програмного забезпечення з використанням цифрового підписування

Виконав: студент 2-го курсу, гр.КІТС-23М
спеціальності 125 – Кібербезпека та захист інформації
Освітня програма – Кібербезпека інформаційних технологій та систем
Луканов М. В.
(прізвище та ініціали)

Керівник: к.т.н., доц., доцент каф. МБІС
Карпінець В. В.
(прізвище та ініціали)

«____» _____ 2024 р.

Опонент: к.т.н., доц., доцент каф. ОТ
Черняк О. І.
(прізвище та ініціали)

«____» _____ 2024 р.

Допущено до захисту
Голова секції УБ кафедри МБІС

Юрій Яремчук
"09" грудня 2024р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека та захист інформації
Освітньо-професійна програма – Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ кафедри МБІС
_____ **Юрій Яремчук**
«20» вересня 2024 р.

ЗАВДАННЯ
на магістерську кваліфікаційну роботу студенту
Луканову Максиму Всеволодовичу

1. Тема роботи: Вдосконалення методу автентифікації у системах контролю версій вихідного коду програмного забезпечення з використанням цифрового підписування
   Керівник роботи : к.т.н., доцент кафедри МБІС Карпінець В. В.
   Затверджені наказом вищого навчального закладу від 17 вересня 2024 року № 310.
2. Строк подання студентом роботи – за тиждень до захисту.
3. Вихідні дані до роботи: Наукові праці у сфері криптографії, досліджень автентифікації, вихідний код програмного забезпечення для тестування.
4. Зміст текстової часини: У першому розділі проведено аналіз методів автентифікації та визначено їх вразливості. У другому розділі здійснено проєктування алгоритму автентифікації на основі цифрового підписування, а також проведено порівняльний аналіз з іншими методами автентифікації на стійкість до загроз. Третій розділ присвячений практичній реалізації програмного забезпечення. У четвертому розділі обґрунтовано економічну ефективність запропонованого програмного забезпечення.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень). У першому розділі магістерської кваліфікаційної роботи наявно 8 рисунків, у другому розділі 5 рисунків, у третьому розділі 12 рисунків

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина	Карпінець В. В., доцент каф. МБІС		
Економічна частина	Бурєннікова Н. В., проф. каф. ЕПВМ		

7. Дата видачі завдання 20 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напрямку магістерської кваліфікаційної роботи, формулювання теми	20.09.2024	25.09.2024	
2	Аналіз предметної області обраної теми	26.09.2024	05.10.2024	
3	Розробка алгоритму роботи	06.10.2024	16.10.2024	
4	Написання магістерської кваліфікаційної роботи на основі розробленої теми	17.10.2024	20.11.2024	
5	Передзахист магістерської кваліфікаційної роботи	21.11.2024	25.11.2024	
6	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	26.11.2024	08.12.2024	
7	Захист магістерської кваліфікаційної роботи	10.12.2024	11.12.2024	

Студент _____ Луканов М. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Карпінець В. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.056.52

Луканов М. В. Вдосконалення методу автентифікації у системах контролю версій вихідного коду програмного забезпечення з використанням цифрового підписування. Магістерська кваліфікаційна робота зі спеціальності 125 – «Кібербезпека», освітня програма «Кібербезпека інформаційних технологій та систем». Вінниця: ВНТУ, 2024. 108 с.

На укр. мові. Бібліогр.: 34 назв; рис.: 25; табл. 10.

У магістерській кваліфікаційній роботі представлено вдосконалений метод автентифікацій у системах контролю версій з використанням цифрового підписування.

В першому розділі розглянуто сучасні методи автентифікації у системах контролю версій, проведено аналіз існуючих методів автентифікації та визначено їх вразливості.

У другому розділі здійснено проектування алгоритму автентифікації на основі цифрового підписування, а також проведено порівняльний аналіз з іншими методами автентифікації на стійкість до загроз, а також виконано огляд відповідності міжнародним стандартам з кібербезпеки. Результати аналізу свідчать, про успішність вдосконаленого методу та доцільність його застосування на практиці.

У третьому розділі здійснено практичну реалізацію запропонованого програмного забезпечення та проведено тестування.

У четвертому розділі роботи здійснено аналіз економічної доцільності розробки, який свідчить про її високий комерційний потенціал та доцільність подальшого впровадження.

Ключові слова: методи автентифікації, системи контролю версій, цифровий підпис, кібербезпека

ABSTRACT

UDC 004.056.52

Lukanov M. V. Improving the authentication method in software source code version control systems using digital signature. Master's qualification work in specialty 125 - “Cybersecurity”, educational program “Cybersecurity of information technologies and systems”. Vinnytsia: VNTU, 2024. 108 c.

In Ukrainian. Bibliography: 34 titles; Figures: 25; Table 10.

This master's thesis presents an improved method of authentication in version control systems using digital signatures.

The first chapter discusses modern authentication methods in version control systems, analyzes existing authentication methods and identifies their vulnerabilities.

The second section designs an authentication algorithm based on digital signatures, conducts a comparative analysis with other authentication methods for threat resistance, and reviews compliance with international cybersecurity standards. The results of the analysis indicate that the improved method is successful and can be applied in practice.

In the third section, the practical implementation of the proposed software is carried out and tested.

The fourth section of the paper analyzes the economic feasibility of the development, which indicates its high commercial potential and the feasibility of further implementation.

Keywords: authentication methods, version control systems, digital signature, cybersecurity

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі швидкий розвиток інформаційних технологій вимагає підвищеної уваги до захисту вихідного коду програмного забезпечення. Системи контролю версій (СКВ) стали незамінними інструментами для розробників, які дозволяють ефективно співпрацювати над проєктами, зберігати історію змін та керувати різними версіями файлів. Проте з розвитком технологій також зростає кількість і складність кіберзагроз, що націлені на втручання у роботу таких систем, зміну коду чи викрадення конфіденційної інформації. Зважаючи на це, забезпечення надійної автентифікації та захисту від несанкціонованого доступу набуває критичного значення.

Цифровий підпис є одним із найефективніших способів забезпечення цілісності та автентичності програмного коду в СКВ. Він дозволяє гарантувати, що всі зміни проходять належну перевірку і не були змінені після їх затвердження. Однак існуючі методи автентифікації мають свої обмеження, що створює потребу у вдосконаленні підходів, здатних мінімізувати ризики сучасних кіберзагроз.

Отже, вдосконалення методу автентифікації у системах контролю версій з використанням цифрового підпису є актуальним і перспективним напрямом дослідження, який сприятиме підвищенню рівня захищеності програмного коду та мінімізуватиме ризики, пов'язані з кібератаками.

Метою даної магістерської роботи є розробка та впровадження удосконаленого методу автентифікації у системах контролю версій програмного забезпечення, що забезпечить ефективний захист від несанкціонованого доступу та збереже цілісність програмного коду.

Для досягнення цієї мети були поставлені наступні завдання:

- Провести аналіз існуючих методів автентифікації у системах контролю версій.
- Оцінити вразливості сучасних СКВ та виклики, що виникають у процесі автентифікації.
- Дослідити можливості застосування цифрового підпису у СКВ для підвищення безпеки.

- Спроектувати та реалізувати алгоритм автентифікації з використанням цифрового підпису.
- Оцінити ефективність запропонованого методу на прикладах та провести його тестування.
- Розробити рекомендації для впровадження удосконаленого методу в системи контролю версій.

Об'єктом дослідження в даній дипломній роботі є процес автентифікації в системах контролю версій вихідного коду.

Предметом дослідження виступають методи і засоби автентифікації з використанням цифрового підпису у СКВ.

Новизна роботи полягає у розробці методу автентифікації, що поєднує надійність цифрового підпису з ефективними криптографічними алгоритмами для захисту вихідного коду та дозволяє ефективно підтверджувати його цілісність.

Практична цінність роботи полягає у створенні рішення, яке сприятиме захисту програмного забезпечення в сучасних умовах, запобігаючи спробам несанкціонованих змін коду та підтримуючи цілісність інформаційних систем.

Робота апробована на конференції “Молодь в науці 2025” [1].

1 ЗАГАЛЬНИЙ АНАЛІЗ ТЕХНОЛОГІЙ АВТЕНТИФІКАЦІЇ В СИСТЕМАХ КОНТРОЛЮ ВЕРСІЙ

1.1 Актуальність дослідження обраної галузі

У сучасному світі, де розвиток інформаційних технологій відбувається з надзвичайно високою швидкістю, контроль версій програмного забезпечення став невід'ємною частиною процесу розробки.

Система контролю версій (СКВ від англ. Version Control System, VCS) – це спеціалізоване місце зберігання коду, система, що записує зміни у файл, чи набір файлів протягом часу і дозволяє повернутися пізніше до певної версії [2].

Системи контролю версій – це набори інструментів, призначені для відстеження змін у програмному коді або інших наборах інформації та управління різними версіями файлів. Вони дозволяють розробникам працювати разом, паралельно вносячи зміни в код або документи, а також відновлювати попередні версії файлів, що є надзвичайно корисним для управління комплексними проєктами [3].

Оскільки великі обсяги конфіденційних даних зберігаються в електронному вигляді, потреба в захисті наших інформаційних ресурсів ніколи не була такою великою. Кіберзагрози розвиваються щодня, вимагаючи все більш суворих заходів безпеки, а ключі та прості паролі вже не справляються з цим завданням. На сьогоднішній день кожній організації потрібна надійна система контролю доступу, яка допоможе захистити конфіденційні дані, зменшити витрати на адміністрування та убезпечити клієнтів і співробітників [4].

Системи контролю версій є критично важливими інструментами в сучасній розробці програмного забезпечення. За допомогою таких систем, команда розробників має можливість ефективно співпрацювати, зберігаючи історію змін та забезпечуючи зворотність будь-яких модифікацій коду. У великих проєктах, де над одним кодом можуть працювати десятки або навіть сотні розробників, системи контролю версій стають незамінними для підтримки організованості та

структурованості процесів розробки. Завдяки цим системам можна уникнути конфліктів у змінах, відстежувати авторів кожної зміни та проводити аудит коду. Це також сприяє швидкому розв'язанню проблем, оскільки зберігається повна історія розвитку проєкту, яка дозволяє зрозуміти, коли і чому були внесені конкретні зміни.

Системи контролю версій (СКВ) загалом виконують чотири основні завдання: доступ до коду, журнал змін коду, розгалуження розробки, підтримка версій програмного продукту (рисунок 1.1)

Рисунок 1.1 – Основні завдання систем контролю версій

Розглянемо детальніше кожне із завдань СКВ:

1. Доступ до коду. Вихідний код проєкту зберігається у віддаленому репозиторії, до якого розробники звертаються, щоб завантажити актуальну версію файлів або внести свої зміни. Це створює основу для командної роботи над проєктом.

2. Журнал змін коду. Відстеження внесених змін (комітів) [5] дає змогу бачити, хто, коли та що саме змінював у коді, допомагає вирішувати конфлікти між одночасними модифікаціями та дозволяє повернутися до будь-якого попереднього стану проєкту.

3. Розгалуження розробки. Розробники можуть працювати над новими функціями у своїх гілках, не впливаючи на стабільність основної версії проєкту, що дозволяє незалежно тестувати новий функціонал [6].

4. Підтримка версій програмного продукту. Під час оновлення продукту створюються релізні версії з використанням тегів, які фіксують стан проєкту на момент випуску. Це полегшує подальше тестування та перегляд змін.

На сьогодні СКВ класифікують на три основні типи: локальні, клієнт-серверні (централізовані) та розподілені (децентралізовані). Також системи контролю версій можуть бути відкритими або закритими для користувачів (таблиця 1.1).

Таблиця 1.1 – Класифікація систем контролю версій

Типи СКВ	Локальні системи контролю версій	Клієнт-серверні (централізовані) системи контролю версій	Розподілені (децентралізовані) системи контролю версій
Приклад	RCS	Subversion (SVN), Perforce	GitHub, GitLab, Bitbucket
Використання	Підходить для одного розробника, оскільки історія змін зберігається локально.	Дані зберігаються на центральному сервері, клієнти підключаються для синхронізації.	Кожен користувач має повну копію репозиторію
Переваги	Простота у використанні	Централізований контроль	Незалежність від центрального сервера
Недоліки	Неможливість командної роботи	Залежність від сервера	Потребує більше місця на локальних машинах

Локальні системи контролю версій

Локальні системи контролю версій – це найпростіший і найперший вид систем, який використовується для збереження історії змін файлів. На відміну від більш сучасних централізованих або розподілених СКВ, локальні системи

зберігають всю інформацію на одному комп’ютері. Це означає, що доступ до історії змін або можливість повернення до попередньої версії є лише на конкретному пристрої, де зберігається репозиторій. Такий підхід має обмежену функціональність, проте задовольняє базові потреби для індивідуальних проєктів. Схема роботи такої СКВ наведено на рисунку 1.2.

Основна особливість локальних СКВ – це локальне зберігання змін у файлах. Принцип роботи таких систем полягає у фіксації змін у спеціальній базі даних, яка зазвичай записує модифікації, додавання нових версій та видалення старих. Локальна система створює резервні копії кожної версії файлу, зберігаючи тим самим контроль за його історією. Це допомагає уникати помилок, які часто виникають при ручному копіюванні файлів у різні директорії.

Однією з найвідоміших локальних СКВ є RCS (Revision Control System). RCS зберігає всі версії файлів локально, дозволяючи кожному файлу мати контрольовану історію змін. Ця система корисна для відстеження змін у документах і є доволі простою у використанні, проте її функціональні можливості обмежені, якщо проєкт потребує командної роботи або синхронізації між кількома пристроями [7].

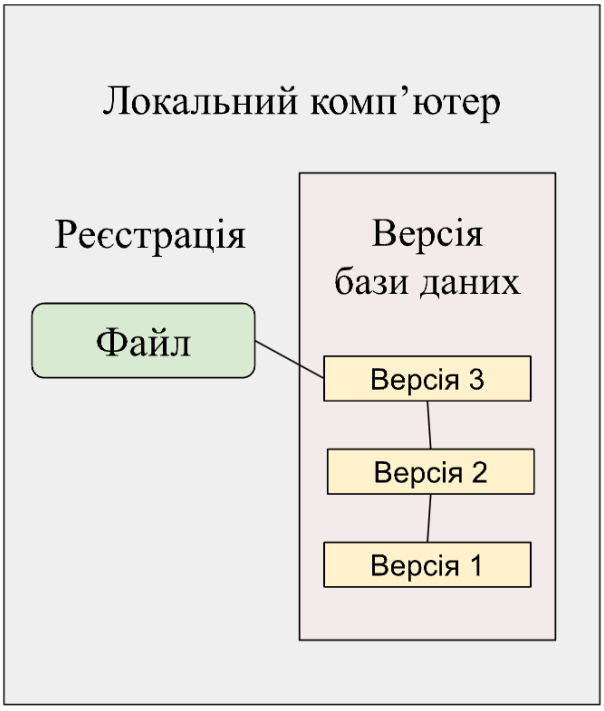


Рисунок 1.2 – Схема роботи локальної системи контролю версій

Хоча локальні СКВ, такі як RCS, зараз використовуються рідко, їхня концепція стала основою для розвитку складніших СКВ, таких як централізовані та розподілені системи. У сучасних проєктах, де потрібна командна робота або віддалений доступ, локальні СКВ поступаються місцем більш досконалим рішенням, як-от Git чи SVN, які дозволяють працювати над одним проєктом команді з будь-якої точки світу.

Централізовані системи контролю версій

Централізовані системи дозволяють кожному розробнику мати доступ до одного і того ж репозиторію, розміщеного на сервері. Користувачі можуть брати останню версію файлів для редагування, вносити свої зміни і завантажувати їх на сервер. Система контролює конфлікти, якщо кілька користувачів редагують одні й ті ж файли, та зберігає хронологію змін. Це дозволяє швидко відновити попередні версії файлів або відстежити, хто вніс конкретні зміни.

Однією з найпоширеніших централізованих СКВ є Subversion (SVN). SVN працює за принципом клієнт-сервер, де всі файли зберігаються на віддаленому сервері, а кожен користувач працює з локальною копією (рисунок 1.3). Іншим прикладом є CVS (Concurrent Versions System), яка також базується на централізованій моделі та підтримує паралельну роботу кількох користувачів над одним проєктом [8].

Попри те, що централізовані системи залишаються популярними, вони поступово поступаються розподіленим СКВ, таким як Git. Розподілені системи надають користувачам більше автономії та дозволяють повноцінно працювати над проєктом навіть без доступу до центрального сервера. Проте централізовані СКВ, такі як SVN, продовжують активно використовуватися в проєктах, де централізоване управління є критично важливим і де потрібен постійний контроль над версіями на одному сервері.

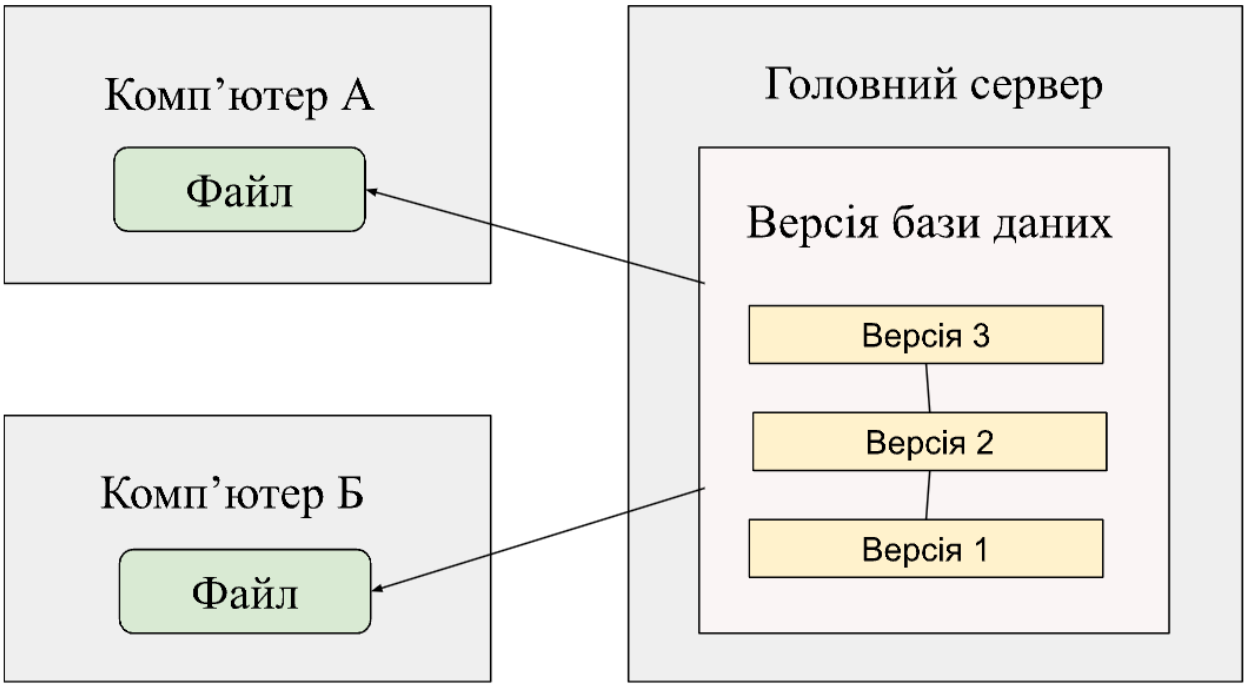


Рисунок 1.3 – Схема роботи централізованої системи контролю версій

Централізовані системи контролю версій були важливим етапом у розвитку СКВ, оскільки забезпечили можливість командної роботи над проєктами, централізованого управління версіями та єдиного сховища даних. Вони досі знаходять застосування в сучасних проєктах завдяки простоті організації та керування доступом, хоча їх поступово замінюють розподілені системи з ширшими можливостями.

Децентралізовані системи контролю версій

Децентралізовані або розподілені системи контролю версій є найсучаснішою і найгнучкішою моделлю серед СКВ. На відміну від централізованих систем, де вся історія змін зберігається на одному сервері, в розподілених системах кожен користувач має повну копію всього репозиторію, включаючи історію всіх версій файлів. Це забезпечує високу стійкість до відмов та гнучкість у роботі, адже кожен розробник може працювати з локальною копією репозиторію, синхронізуючи зміни з іншими лише за необхідності.

У децентралізованих системах, таких як Git, кожен користувач має свою локальну копію репозиторію з повною історією змін, включно з кожним комітом і всіма гілками. Це дозволяє працювати автономно, без підключення до центрального сервера, і дає змогу зберігати локальні коміти, які можна об’єднати з основною гілкою після підключення до мережі. Розподілені системи підтримують створення різних гілок і паралельну розробку, забезпечуючи потужні інструменти для контролю версій навіть у складних проєктах. Схема роботи зображена на рисунку 1.4.

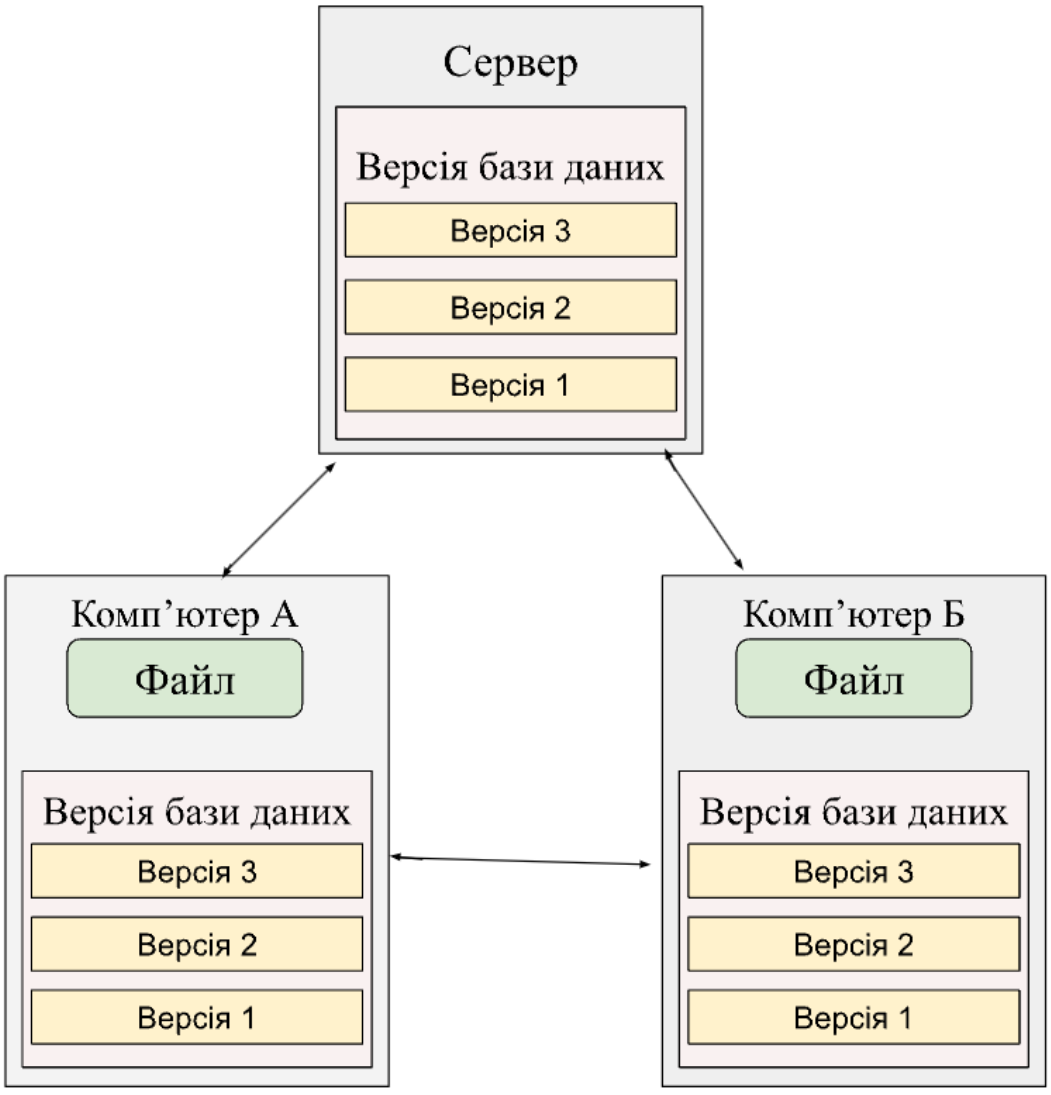


Рисунок 1.4 – Схема роботи децентралізованої системи контролю версій

Розподілені системи контролю версій, зокрема Git, стали стандартом у сучасній розробці програмного забезпечення, особливо для великих проєктів з багатьма учасниками. Вони ідеально підходять для віддаленої командної роботи, забезпечуючи розробникам високий рівень автономії та контроль над власними змінами. Популярні платформи для спільної роботи, такі як GitHub та GitLab, також базуються на розподіленій моделі, надаючи можливість не лише контролювати версії, але й ефективно співпрацювати через систему pull-запитів та перегляду коду [9].

Найпопулярнішими системами контролю версій є саме децентралізовані, серед них такі як GitHub, GitLab, Bitbucket. Серед трьох лідерів СКВ - найбільшою довірою користувачів володіє саме GitHub. Цим сервісом користується безліч користувачів по всьому світу, в тому числі такі відомі компанії як Spotify, The New York Times, Philips, Carlsberg Group, Duolingo, Shopify та інші [10].

Децентралізовані системи контролю версій, такі як Git, забезпечують автономну роботу розробників, високу стійкість до збоїв та гнучкість у створенні і злитті гілок. Вони відкрили нові можливості для командної роботи та стали ключовим інструментом у сучасному розробницькому середовищі. Втім, зі зростанням складності та популярності таких систем з’являється і новий виклик — забезпечення автентичності та безпеки внесених змін.

Оскільки децентралізовані СКВ дозволяють кожному розробнику працювати з повною копією репозиторію та вільно вносити зміни, ризик несанкціонованих модифікацій або помилок зростає. У середовищах із багатьма розробниками критично важливо мати чітку ідентифікацію кожного користувача та забезпечити автентичність комітів. Це дозволяє уникнути випадкового чи зловмисного втручання в код і забезпечує точне відстеження внесених змін. У зв’язку з цим, надійні методи автентифікації стають невід'ємною частиною безпеки децентралізованих систем контролю версій, підвищуючи довіру до змін та захищаючи цілісність проєктів у процесі розробки.

Системи контролю версій вихідного коду програмного забезпечення відіграють ключову роль у процесі розробки програмного забезпечення, ставши

певним стандартом, забезпечуючи збереження історії змін, координацію між розробниками та підтримку цілісності коду. Проте безпека цих систем залишається критичним питанням, особливо з огляду на ризики внесення неавторизованих змін сторонніми особами або під час диверсії. Автентифікація користувачів та контроль цілісності коду є основними заходами, які дозволяють підвищити захист у таких системах [11].

1.2 Виклики та загрози безпеці у процесі автентифікації

Процес автентифікації в системах контролю версій стикається з рядом викликів та загроз безпеці, які можуть поставити під загрозу цілісність і захищеність системи. Однією з основних загроз є крадіжка даних, коли зловмисники можуть отримати доступ до паролів, сертифікатів або інших ключів автентифікації. Найпоширенішими загрозами у процесі автентифікації є фішинг-атаки, парольні атаки, атаки на токени та соціальна інженерія.

Фішинг у системах контролю версій спрямований на введення розробників в оману за допомогою підроблених повідомлень чи веб-сайтів. Зловмисники можуть, наприклад, створити фальшиву сторінку входу до репозиторію, де користувачі вводять свої облікові дані, не підозрюючи, що передають їх шахраям. Метою таких атак є доступ до конфіденційних даних, включаючи пароль чи сесійний токен, які використовуються для доступу до системи [12]. Складність полягає в тому, що фішинг може бути спрямований не тільки на новачків, але й на досвідчених користувачів через дуже реалістичні повідомлення чи підроблені інтерфейси.

Парольні атаки на системи контролю версій охоплюють різноманітні методи, серед яких найпоширеніші – перебір паролів та словниковий підбір пароля. Перебір паролів відбувається, коли зловмисники намагаються випробувати всі можливі комбінації для злому пароля, хоча для складних паролів це вимагає багато часу і ресурсів. Словниковий підбір пароля полягає у використанні поширених слів чи фраз для підбору пароля, що може бути ефективним, якщо користувачі обирають прості паролі. Обидва методи можуть становити серйозну загрозу для репозиторіїв, якщо система контролю версій не має додаткових заходів безпеки, таких як

обмеження на кількість спроб введення пароля або використання двофакторної автентифікації [13].

Атаки на токени націлені на перехоплення або несанкціоноване використання токенів сесії, які застосовуються для ідентифікації користувача після входу в систему. Токени сесії дозволяють залишатися авторизованим на деякий час без повторного введення пароля, що робить їх зручними, але водночас вразливими. Якщо зловмисник отримає доступ до токену через перехоплення чи інші методи, він може скористатися ним для доступу до облікового запису користувача. Такі атаки є серйозною загрозою, оскільки надають шахраям можливість обійти автентифікацію і безперешкодно отримати доступ до файлів чи коду в репозиторії.

Соціальна інженерія – це один з найефективніших способів атаки через несвідомість людей з доступом до конфіденційної інформації. Цей метод використовує маніпуляцію та переконання для обману користувачів для їхньої інформаційної безпеки чи облікових даних. Найчастіші прийоми соціальної інженерії: маніпуляція, обман, психологічний тиск, підробка особистості [14] .У таблиці 1.2 наведено перелік рішень, спрямованих на посилення захисту автентифікації в системах контролю версій.

Таблиця 1.2 – Перелік рішень, спрямованих на посилення захисту автентифікації

Типи атаки	Рішення
Фішинг	Навчання працівників Використання антифішингових фільтрів 2FA Перевірка URL-адрес
Парольні атаки	Використання парольних менеджерів Надійні паролі
Атаки на токени	Одноразові токени Використання протоколу HTTPS
Соціальна інженерія	Навчання працівників Постійний аудит

Для забезпечення стійкості систем до таких загроз необхідно застосовувати комплексний підхід до безпеки, що включає сучасні рішення для захисту від різноманітних типів атак. Найважливіше, що можна зробити для зменшення ризику порушення безпеки – увімкнути багатофакторну автентифікацію й вимагати від користувачів надавати принаймні два фрагменти інформації. Зловмисникам набагато складніше викрадати дані, якщо ви використовуєте кілька способів автентифікації [15]. Однак у випадку фішингу двофакторна автентифікація не здатна надати повний захист, оскільки зловмисники викрадають одночасні паролі також.

1.3 Аналіз сучасних методів автентифікації у системах контролю версій

Автентифікація являє собою процедуру перевірки дійсності ідентифікаторів. У процесі ідентифікації та автентифікації користувач або програма (претендент) запитує доступ у системи (верифікатора). Спочатку здійснюється ідентифікація. Верифікатор жадає від претендента пред’явити деякий ідентифікатор і перевіряє приналежність пред’явленого ідентифікатора, безлічі зареєстрованих у системі. У випадку коректності ідентифікатора, верифікатор виконує процедуру автентифікації (наприклад, запитує пароль), щоб переконатися, що претендент є саме тим, за кого себе видає. Допуск претендента в систему дозволяється тільки у випадку успішного завершення процедури автентифікації [11].

Потреба у використанні якісних методів автентифікації у системах контролю версій (СКВ) є дуже вираженою в умовах зростаючої кількості кіберзагроз, коли хакери намагаються отримати несанкціонований доступ до репозиторіїв, змінюючи код або викрадаючи конфіденційну інформацію, особлива увага до безпеки повинна приділятись у випадках, коли використовується автоматичне розгортання або оновлення додатків на серверах. Це підвищує ризики для компаній, оскільки програмний код часто містить унікальні алгоритми та технології, що становлять інтелектуальну власність. Надійні методи автентифікації не лише захищають цю цінність, але й підтримують ефективну командну роботу, оскільки в сучасних

проєктах часто бере участь багато розробників. Чітка автентифікація кожного учасника дозволяє відстежувати зміни і запобігати конфліктам. Крім того, регуляторні вимоги в багатьох галузях зобов'язують дотримуватись високих стандартів безпеки даних серії ISO 2700x, що включає належні методи автентифікації [16].

Існує досить багато методів автентифікації у системах контролю версій, серед них такі:

- Паролі
- Двофакторна автентифікація (2FA)
- SSH-ключі
- Токени доступу

Пароль

Пароль є одним із найпоширеніших і найстаріших методів автентифікації, що використовується у системах контролю версій. Цей метод базується на наданні користувачем певного секретного слова або фрази для підтвердження своєї особи та доступу до системи. У системах контролю версій пароль забезпечує початковий рівень захисту, дозволяючи обмежити доступ до репозиторію або певних дій у ньому лише для авторизованих користувачів.

Цей засіб є найпоширенішим через його простоту реалізації, мізерний об'єм даних якими користувач ділиться з веб-застосунком, а також за малу відповідальність за особисті дані користувача, оскільки зберігати їх надзвичайно просто та безпечно [17] .

На рис. 1.5 зображено схему автентифікації на основі паролю [18].

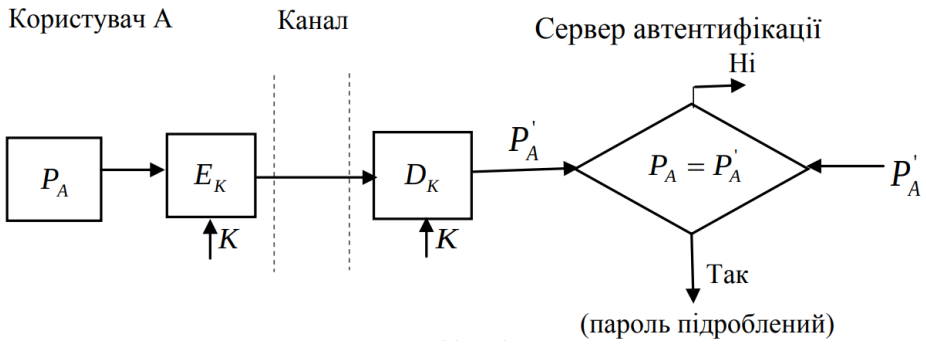


Рисунок 1.5 – Схема автентифікації на основі паролю

Суть парольної автентифікації полягає в тому, що кожен зареєстрований користувач певної системи одержує набір персональних реквізитів (зазвичай використовуються пари логін-пароль). Далі при кожній спробі входу він повинен вказати свою інформацію. Оскільки вона унікальна для кожного користувача, то на підставі її система й робить висновок про особистість та ідентифікує її. Процедуру автентифікації користувача в мережі можна представити наступним чином. При спробі входу в мережу користувач набирає свої логін та пароль. Ці дані надходять і за логіном користувача знаходиться відповідний запис. З нього отримується пароль і порівнюється з тим паролем, який ввів користувач. Якщо вони збіглись, то автентифікація відбулась успішно – користувач отримує ті права та ресурси мережі, які визначені для його статусу системної авторизації [19].

Основним недоліком парольної автентифікації є велика залежність її надійності від самих користувачів, а саме – від їх вибору паролів. Крім того, без застосування додаткових механізмів захисту, парольна автентифікація сама по собі не є достатньо надійною і не може гарантувати необхідного рівня безпеки. У контексті командної роботи над програмними проєктами це становить певний ризик, адже випадкове або зловмисне розголошення пароля може призвести до несанкціонованих змін у репозиторії. Тому паролі часто комбінують з іншими, більш надійними методами автентифікації, зокрема двофакторною автентифікацією або SSH-ключами, щоб покращити безпеку доступу та забезпечити точну ідентифікацію кожного розробника.

Двофакторна автентифікація (2FA)

Двофакторна автентифікація (2FA) є потужним методом підвищення безпеки у системах контролю версій. Вона забезпечує додатковий рівень захисту, який вимагає від користувача, окрім введення пароля, надати ще один фактор підтвердження своєї особи. Зазвичай це може бути одноразовий код, отриманий через SMS, електронну пошту, спеціальний додаток на телефоні або згенерований апаратним токеном.

Отже, у більшості випадків для забезпечення безпечного доступу одного фактору аутентифікації недостатньо. Тому побудувати захищену на 100% систему неможливо. Однак, використовуючи переваги факторів аутентифікації в комплексі, можна звести ризики до мінімуму [20].

Застосування 2FA значно знижує ризик несанкціонованого доступу до репозиторію навіть у разі компрометації пароля. Це критично важливо в умовах, коли багато розробників працюють віддалено або на відкритих платформах, де безпека коду має вирішальне значення. Двофакторна автентифікація дозволяє точніше контролювати доступ і забезпечує надійний рівень автентичності користувачів, що мінімізує ризики випадкових чи зловмисних змін у коді.

Розглянемо сильні й слабкі сторони двофакторної автентифікації. До переваг можна віднести її здатність захистити інформацію від зовнішніх і внутрішніх загроз, наявних при використанні парольної або однофакторної автентифікації. Додавання другого чинника значно мінімізує вірогідність злому: шанси виникнення ситуації, при якій зловмисник окрім логіна і пароля також оволодів іншим чинником, досить малі, і такий спосіб авторизації є досить надійним. При цьому для забезпечення такого захисту необхідно використовувати додаткові програмно-апаратні комплекси, облаштування зчитування даних, їх зберігання, що вимагає певних витрат на їх придбання, впровадження і обслуговування. Безумовно, для того, щоб автентифікація була насправді надійною, важлива не кількість типів ознак, а й якість реалізації механізму на обох сторонах взаємодії як

в частині, що знаходиться у користувача, так і в частині, що знаходиться у перевіряючої сторони. Тому, особливу увагу треба приділити способу зберігання ідентифікаційних даних [21].

У системах контролю версій для двофакторної автентифікації (2FA) використовують кілька ключових методів. Один із найпоширеніших – це одноразові паролі (OTP), які генеруються додатками, такими як Google Authenticator або Authy, і діють протягом короткого часу. Інший варіант – коди, надіслані через SMS, які хоч і менш надійні, але забезпечують базовий захист. Апаратні токени, такі як YubiKey, також часто використовуються, забезпечуючи доступ до системи лише за фізичної наявності пристрою. Ще один метод – біометрична автентифікація, що базується на унікальних характеристиках користувача, таких як відбитки пальців чи розпізнавання обличчя, забезпечуючи високий рівень безпеки. Кожен із цих методів додає додатковий рівень захисту, значно знижуючи ймовірність несанкціонованого доступу до репозиторію.

SSH-ключі

Ключ SSH – це облікові дані безпечного доступу, які використовуються в протоколі Secure Shell (SSH). Пари ключів SSH використовують технологію інфраструктури відкритих ключів (PKI), золотого стандарту цифрової автентифікації та шифрування, щоб забезпечити безпечний і масштабований метод автентифікації [22].

Оскільки протокол SSH широко використовується для зв’язку в хмарних службах, мережевих середовищах, інструментах передачі файлів, інструментах керування конфігурацією та інших залежних від комп’ютера службах, більшість організацій використовують цей тип автентифікації на основі ключів для перевірки ідентичності та захисту цих служб від ненавмисного використання або зловмисних атак.

Пара ключів SSH використовується для автентифікації користувача або процесу, який хоче отримати доступ до віддаленої системи за допомогою

протоколу SSH. Відкритий ключ використовується як користувачем, так і віддаленим сервером для шифрування повідомлень. На стороні віддаленого сервера він зберігається у файлі відкритого ключа. На стороні користувача він зберігається в програмному забезпеченні керування ключами SSH або у файлі на його комп'ютері. Закритий ключ залишається лише в системі, яка використовується для доступу до віддаленого сервера, і використовується для розшифровки повідомлень.

Коли користувач або процес запитує підключення до віддаленого сервера за допомогою клієнта SSH, для завершення автентифікації ініціюється послідовність запитів-відповідей. Сервер SSH розпізнає, що надійшов запит на підключення, і надсилає зашифрований запит виклику, використовуючи спільну інформацію відкритого ключа. Потім клієнт SSH розшифровує повідомлення виклику та відповідає серверу. Користувач або процес повинні правильно відповісти на виклик, щоб отримати доступ. Ця послідовність запитів-відповідей відбувається автоматично між SSH-клієнтом і сервером без будь-яких ручних дій з боку користувача (рисунок 1.6) [23].

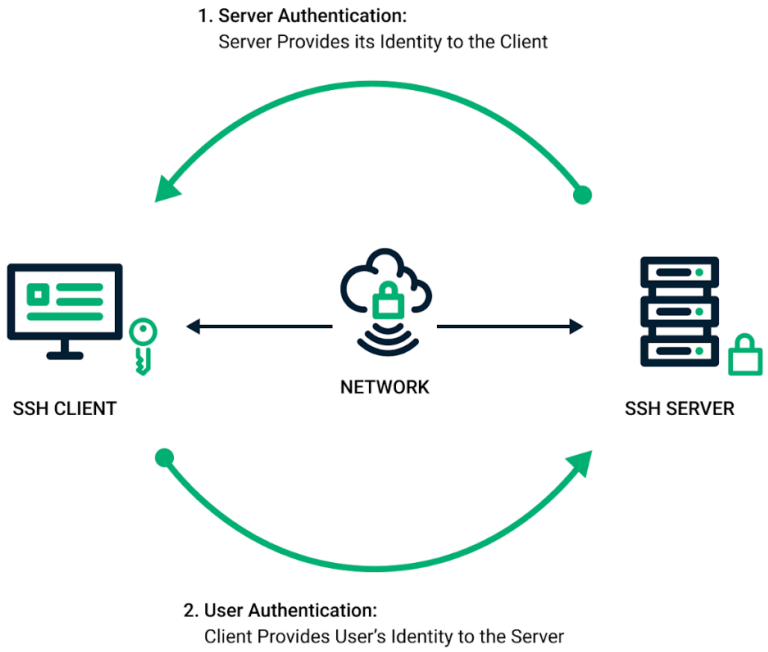


Рисунок 1.6 – Схема роботи SSH-ключів як методу автентифікації

Токени доступу

Такий спосіб автентифікації найчастіше застосовується при побудові розподілених систем Single Sign-On (SSO), де один додаток (service provider або relying party) делегує функцію автентифікації користувачів іншому додатку (identity provider або authentication service). Типовий приклад цього способу — вхід в програму через акаунт у соціальних мережах. Тут соціальні мережі є сервісами автентифікації, а додаток довіряє функцію аутентифікації користувачів соціальних мереж [24].

Токени доступу – це метод автентифікації, який використовується для підтвердження особи користувача та забезпечення безпечного доступу до ресурсів системи без необхідності передачі пароля. Коли користувач вводить свої облікові дані, система перевіряє їх і видає токен, який потім використовується для ідентифікації користувача під час доступу до ресурсів (рис. 1.7). Токени можуть мати обмежений термін дії, що підвищує безпеку, оскільки навіть якщо токен буде викрадений, зловмисник зможе використовувати його лише протягом короткого часу. Важливо забезпечити надійне зберігання токенів на пристроях користувачів, адже їх компрометація може призвести до несанкціонованого доступу. Також, короткий термін дії токенів може створювати певні незручності для користувачів, які вимушені повторно проходити процес автентифікації.

Рисунок 1.7 – Схема роботи токенів доступу як методу автентифікації

Система єдиного входу (Single sign-on, SSO) дозволяє забезпечити безпеку користувачам, застосовуючи один набір облікових даних, щоб отримати доступ до кількох програм або сайтів. Для цього користувачам створюють обліковий запис ідентифікаційної інформації (IdP). IdP перевіряє, чи користувач перевірив дані входу через cookie, проте необхідно затратити більше часу, щоб здійснити налаштування та доступ до різних програм та сайтів через SSO. У системі єдиного входу ідентифікаційні дані набувають форми токенів, що містять ідентифікаційні значення інформації про користувача. Провайдер послуг відправляє токен, в якому міститься інформація про користувача (email або ім’я користувача) системі SSO як частина запиту на автентифікацію користувача. У процесі автентифікації необхідним є забезпечення надійного надсилання даних зв’язку між клієнтом та віддаленим сервером [25].

1.4 Особливості застосування цифрового підпису у системах контролю версій

Електронний цифровий підпис (ЕЦП) – вид електронного підпису, отриманого за результатом криптографічного перетворення набору електронних даних, який додається до цього набору або логічно з ним поєднується і дає змогу підтвердити його цілісність та ідентифікувати підписувача. Електронний цифровий підпис накладається за допомогою особистого ключа та перевіряється за допомогою відкритого ключа [26]. Схематично це зображено на рисунку 1.8.

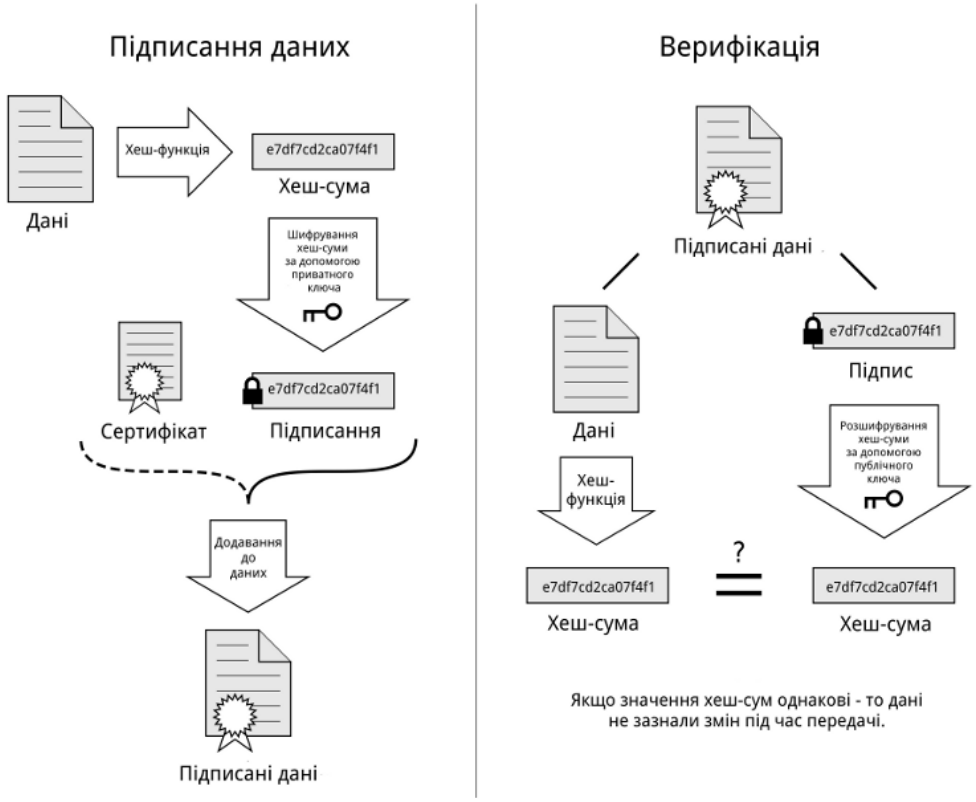


Рисунок 1.8 – Схема цифрового підписування

В Україні правовий статус і використання ЕЦП визначаються Законами «Про електронні документи та електронний документообіг» та «Про електронний цифровий підпис» [27, 28].

Цифровий підпис у системах контролю версій забезпечує високий рівень автентифікації та цілісності даних. Завдяки криптографічним алгоритмам, цифровий підпис дозволяє підтвердити авторство та справжність внесених змін, що є особливо важливим для спільних проєктів, де код модифікується багатьма розробниками. Підпис кожного коміту гарантує, що зміни походять від авторизованого користувача та не були підроблені чи змінені після їхнього створення, однак в існуючому варіанті підпису в СКВ не надається можливость централізованого керування та перевірки коду, а також будь-яка особа з правами адміністратора здатна внести зміни в обхід даної системи.

1.5 Висновки та постановка задач

У даному розділі було виконано всебічний аналіз технологій автентифікації, що використовуються в системах контролю версій, з акцентом на підвищення безпеки та надійності автентифікації користувачів у цих системах. Аналіз було розпочато з вивчення актуальності дослідження, яке базується на зростаючій потребі в захисті даних та управлінні доступом до спільних репозиторіїв. Сучасні системи контролю версій, незалежно від їхнього типу – локальні, централізовані або децентралізовані – відіграють важливу роль у координації командної роботи, підтримуючи історію змін і забезпечуючи можливості для ефективного збереження різних версій програмного коду. Розвиток цих систем викликав необхідність у надійних методах автентифікації, що мінімізують ризики несанкціонованого доступу.

Дослідження сучасних методів автентифікації показало, що традиційні підходи, такі як паролі та навіть двофакторна автентифікація, часто не забезпечують необхідного рівня безпеки для захисту від кібератак, зокрема

фішингу, атак на токени, перехоплення сесій та соціальної інженерії. Традиційна автентифікація покладається на користувача у створенні надійних паролів, що не завжди виправдовує себе через ризик людських помилок. Альтернативні методи, такі як двофакторна автентифікація, забезпечують більш надійний захист, але й мають свої обмеження та є вразливими до деяких типів атак.

Застосування цифрового підпису у СКВ продемонстровано як один із найефективніших способів забезпечення автентичності змін у коді та захисту від підробок і несанкціонованих модифікацій. Використання цифрових підписів дозволяє гарантувати, що всі зміни в коді проходять перевірки регулюючої особи, що знижує ймовірність маніпуляцій та зміцнює загальну довіру до збережених даних.

На основі виконаного аналізу можна зробити висновок, що впровадження нових і вдосконалення методів автентифікації, таких як цифровий підпис у поєднанні з двофакторною автентифікацією, здатне значно підвищити безпеку систем контролю версій. Таке комплексне вирішення проблеми автентифікації дозволить не лише захистити дані від несанкціонованого доступу, а й зміцнить довіру до системи загалом, зберігаючи цілісність даних та авторизацію користувачів на високому рівні.

2 ВДОСКОНАЛЕННЯ МЕТОДУ АВТЕНТИФІКАЦІЇ З ВИКОРИСТАННЯМ ЦИФРОВОГО ПІДПИСУ

2.1 Проєктування алгоритму автентифікації на основі цифрового підпису

Опишемо алгоритм автентифікації вихідного коду програмного забезпечення з використанням цифрового підпису файлів у системах контролю версій. Даний підхід спрямований на забезпечення цілісності та автентичності всіх файлів проєкту і можливості відстеження авторства, а використовується для перевірки та підтвердження безпечності та надійності коду відповідними регулюючими особами, підрозділами або органами.
Цей підхід гарантує, що тільки перевірений код, підтверджений уповноваженими особами, отримує цифровий підпис та вважається готовим для подальшого використання у вигляді розміщення на серверах, інших кінцевих точках, поширення серед клієнтів тощо.

Першим етапом даного алгоритму є генерація пари ключів (відкритого та закритого) відповідними регулюючими особами та забезпечення належного зберігання закритого ключа, враховуючи необхідні рівні захисту для дотримання конфіденційності, цілісності та доступності даних ключів відповідно до вимог організації, державних законів та міжнародних стандартів з кібербезпеки [29].

Наступним кроком є розміщення відкритого ключа в централізованій базі для подальшого використання під час верифікації кінцевим користувачем або автоматичними системами розгортання або оновлення додатків. На даному етапі забезпечується розмежування прав на підпис файлів - вказані права надаються лише відповідним особам, таким як керівники відділів, команд, фахівців з інформаційної безпеки, згідно принципу PoLP [30].

Вказані етапи додатково зображені на блок схемі (рис. 2.1).

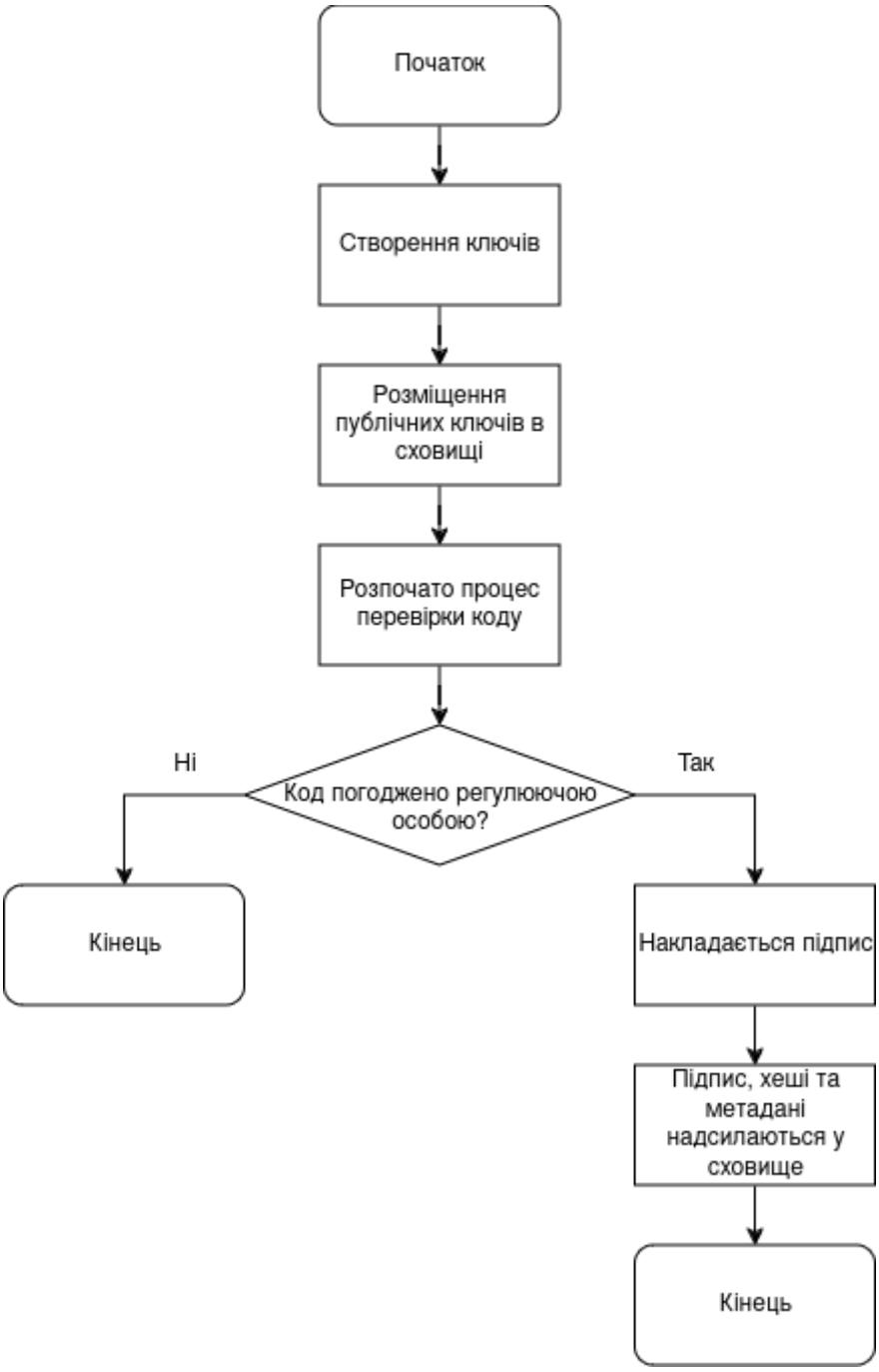


Рисунок 2.1 – Блок-схема алгоритму підпису файлів

На певному етапі розробки програмного продукту, залежно від потреб організації це може бути розширення функціональності програмного забезпечення, виправлення помилок, вразливостей або навіть кожна зміна в коді додатка, регулююча особа виконує перевірку розроблювального програмного забезпечення та, у разі погодження внесених змін, створює хеш кожного файлу, який має бути підписаний і генерує цифровий підпис, з використанням особистого приватного ключа. Збереження підписів, хешів та метаданих файлів забезпечується у

централізованому надійному сховищі, що має обмежений доступ на редагування даних, та оновлюється лише у випадку підпису файлів. Метадані містять інформацію про автора змін (розробника), особу, що підписала файл, хеш та дату підписання. Це дозволяє відстежити кожний етап перевірки й схвалення змін. Додатково даний процес зображено на блок схемі (рис. 2.2)

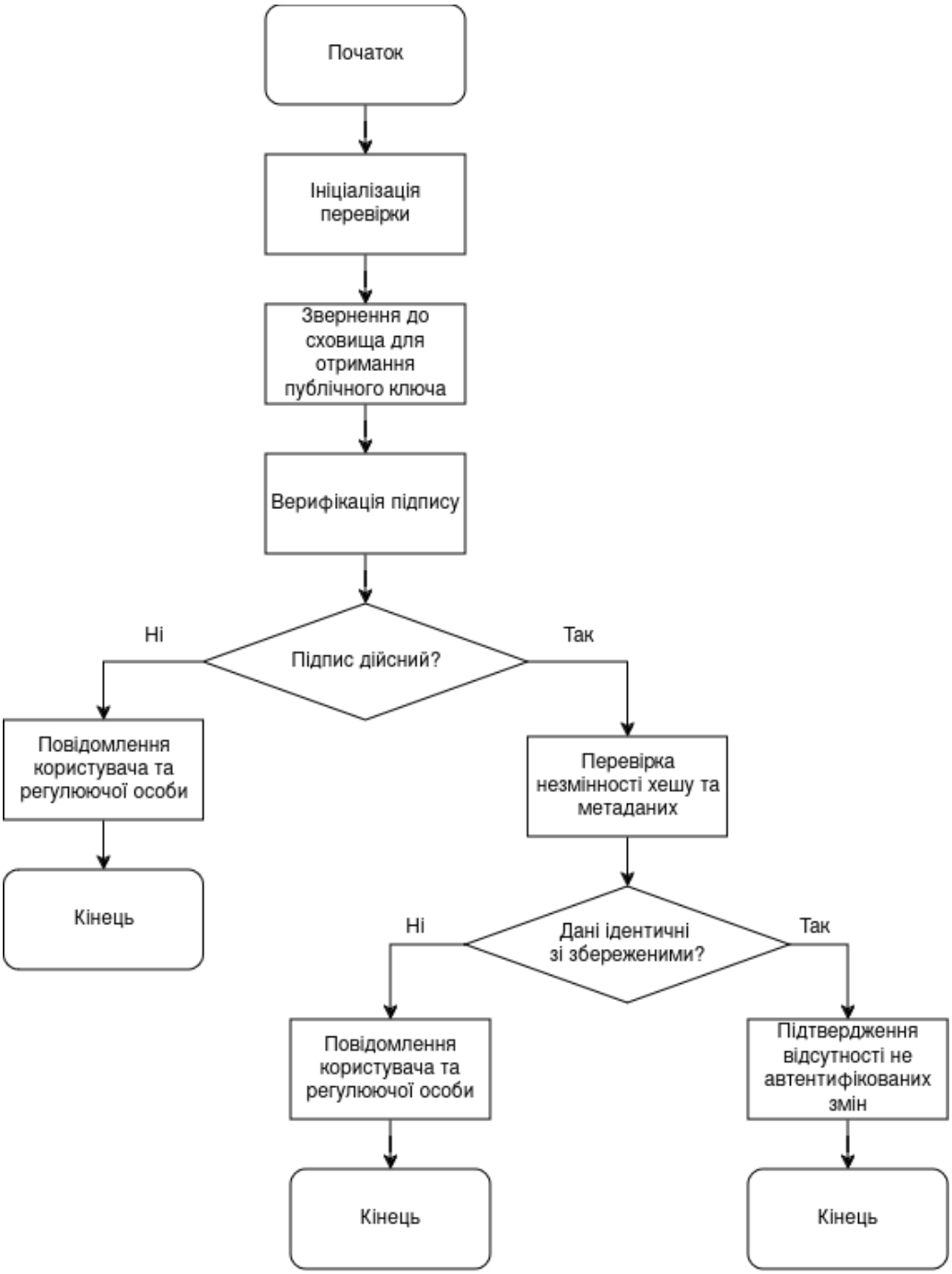


Рисунок 2.2 – Блок-схема алгоритму перевірки підпису

Перевірка автентичності може бути проініціалізована на вимогу, наприклад - у випадку підозри на несанкціонованої зміни, виникнення інциденту, пов’язаного з кодом програмного продукту та його функціональності тощо. Також перевірка відбувається автоматично у випадку використання комбінованої практики безперервної інтеграції та безперервної доставки, безперервного розгортання програмного продукту. Даний процес виглядатиме наступним чином - виконується процес верифікації, який звіряє хеші файлів з їх підписами, використовуючи публічні ключі з централізованого сховища, додатковим аспектом перевірки є звірка метаданих з тими, які були збережені на етапі підпису. Якщо підпис верифікується, а метадані залишились без змін - результат перевірки вважається успішним, що дає змогу підтвердити відсутність неавтентифікованих змін у коді. У випадку використання автоматизованих систем безперервної інтеграції, доставки та розгортання програмного продукту - успішна перевірка дозволяє розпочати даний процес. Якщо підпис виявляється недійсним або ж метадані файлів було змінено - про це повідомляється особі яка ініціалізувала перевірку, надсилається повідомлення регулюючому органу, а у випадку використання автоматизованого процесу - його зупинка.

2.2 Розробка алгоритму роботи програмного додатку

Опишемо основні етапи роботи розроблювального програмного додатка, який забезпечить додаткову надійність систем контролю версій вихідного коду програмного забезпечення за допомогою використання цифрового підписування.

Першим етапом є забезпечення можливості зручної, інтуїтивно зрозумілої, а головне - безпечної можливості генерації ключів та поширення відкритого ключа регулюючими особами. Для цього буде створено скрипт, який, для зручності, при запуску запитує ім’я та прізвище особи, яка генерує ключ. Після введення вказаних даних буде запропоновано згенерувати новий ключ та для продовження необхідно вказати шлях, за яким зберегти приватний ключ. При залишенні даного поля пустим, ключ буде збережено в поточній директорії. Після генерації пари ключів

приватний буде збережено у вказані директорії, а публічний буде надіслано адміністратору для внесення у централізоване сховище. Оскільки ця процедура є разовою для кожного підписанта, дані роботи не потребують великої кількості часу, а підтвердження коректності отриманого ключа можна виконати використавши альтернативні шляхи комунікацій з власником приватного ключа, підписанням тестового файлу тощо, якщо така необхідність буде існувати і відповідно до політик, інструкцій та інших регламентуючих документів, які можуть існувати в організації, яка використовуватиме дане програмне забезпечення. Алгоритм роботи даної частини додатка зображено на блок схемі (рис 2.3).

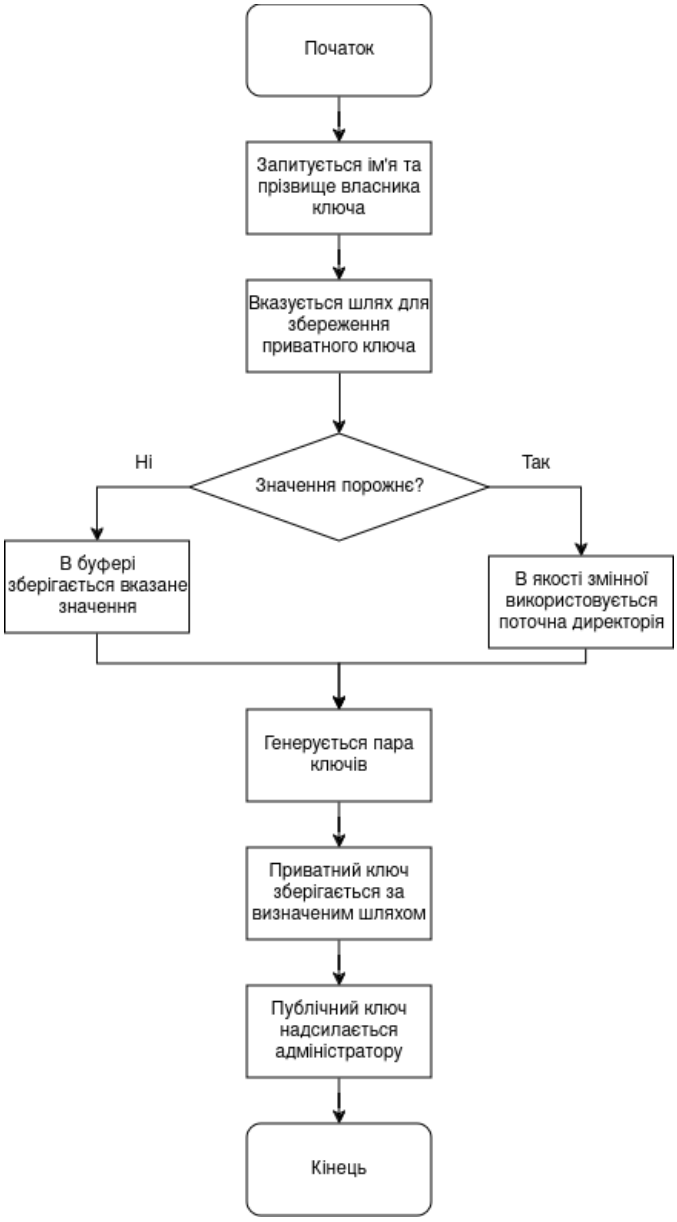


Рисунок 2.3 – Блок-схема роботи скрипта для генерації ключів

Наступною складовою функціонування програмного додатка є підпис перевірених файлів регулюючою особою. Для ознайомлення і тестування перевіряючий завантажує файли проєкту та після виконання вказаних робіт запускає застосунок для підпису. Першим етапом підпису є запитування шляху до приватного ключа. Якщо цей шлях є коректним, запитується шлях до файлів проєкту, якщо шлях порожній - використовується поточна директорія та створюється цифровий підпис кожного файлу, зберігаються їхні хеш-суми, після чого підписи, хеші та метадані файлів надсилаються на централізоване сховище. Перед записом надісланої інформації у базу даних виконується перевірка відповідності підпису з відкритим ключем і у випадку актуальності - дані зберігаються у сховищі. Варто зазначити, що даний процес є автоматизованим та не потребує додаткових дій зі сторони адміністратора сховища, що мінімалізує людські часовитрати. Додатково цей процес зображено на блок схемі (рис. 2.4).

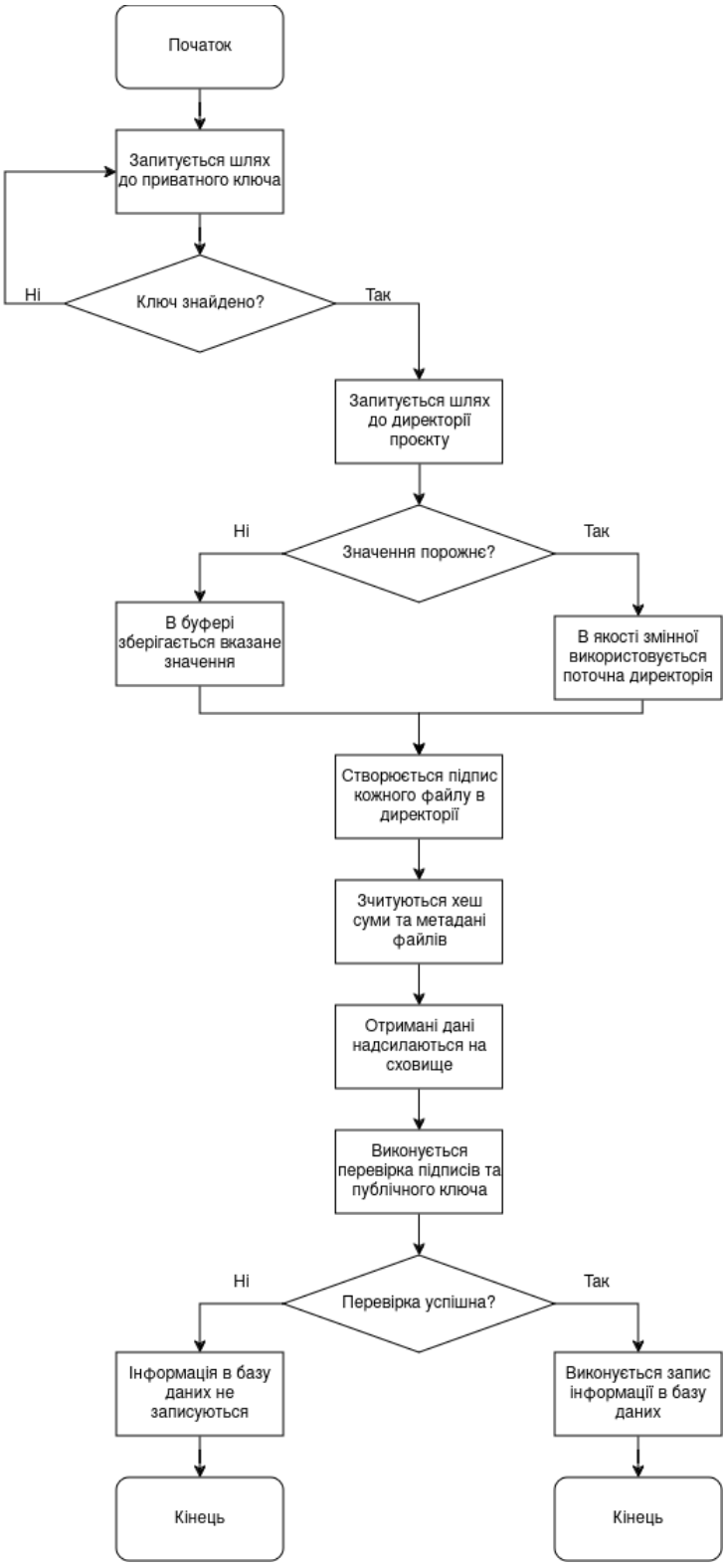


Рисунок 2.4 – Блок-схема роботи додатка під час накладання підпису

Процес перевірки підпису може виконуватись як в автоматичному режимі, так і при ініціалізації зі сторони людини, алгоритм роботи буде мати аналогічний характер та відрізнятиметься лише кінцевими пунктами. Першим етапом є

ініціалізація перевірки, при якій запускається програма верифікація. В даному коді буде створено процес звернення до централізованого сховища для отримання публічного ключа, підписів та метаданих файлів. В першу чергу, обраховується хеш поточних даних та звіряється підпис за допомогою публічного ключа.У випадку, якщо підпис дійсний - перевірка вважається успішною, якщо ж ні – надсилається повідомлення про інцидент відповідальній особі. У випадку, якщо перевірка використовується для автоматизованих процесів безперервної інтеграції, доставки та розгортання програмного продукту системі в контролю версій вихідного коду програмного забезпечення – виконується даний автоматизований процес. Додатково даний алгоритм зображено на блок схемі (рис. 2.5)

Рисунок 2.5 – Блок-схема роботи додатка під час перевірки підпису

2.3 Аналіз рівня захисту системи контролю версій з використанням вдосконаленого методу автентифікації та відповідність міжнародним стандартам

Впровадження вдосконаленого методу автентифікації на основі цифрового підпису у системи контролю версій значно підвищує рівень безпеки, оскільки створює механізм погодження змін та покращує захист від ряду атак.

Для детального порівняння рівня захисту з існуючими мезанізмами автентифікації в таблиці 2.1 наведено огляд захищенності від таких загроз як несанкціонований доступ через викрадені облікові дані, підробка змін у коді, компрометація ключів/токенів, недосконалості розробленого коду та соціальної інженерії.

Таблиця 2.1 – Огляд методів автентифікації

	Пароль	Двофакторна автентифікація	SSH-ключі	Токени доступу	Вдосконалений метод з цифровим підписом
Несанкціонований доступ через викрадені облікові дані	Захист слабкий, можливе використання скомпрометованих паролів.	Стійкий до компрометації паролів завдяки додатковому фактору.	Високий рівень захисту, якщо приватний ключ зберігається безпечно.	Високий рівень, якщо токен зберігається у захищеному середовищі.	Забезпечуєтся через обов’язковий процес перевірки підпису.
Підробка змін у коді	Не захищає.	Не захищає.	Не захищає.	Не захищає.	Підпис кожного файлу забезпечує виявлення змін.
Компрометація ключів/токенів	Немає ключів або токенів.	Немає ключів або токенів.	Уразливий, якщо приватний ключ зкомпрометовано.	Уразливий, якщо токен вкрадено.	Забезпечує додатковий фактор захисту. Ключ підпису не надає доступ до репозиторію

Продовження таблиці 2.1					
Недосконалості розробленого коду	Не захищає.	Не захищає.	Не захищає.	Не захищає.	Регулююча особа схвалює зміни перед підписом.
Соціальна інженерія	Захист слабкий.	Захист середній, через фішинговий сайт можливо отримати код з другого фактору	Уразливий, якщо приватний ключ не захищений.	Уразливий, якщо токен отримано зловмисником.	Приватний ключ не поширюється підписантом та використовується лише у визначених програмах

На основі даного аналізу можна зробити висновки, що розроблювальний вдосконалений метод автентифікації значно покращує рівень захисту та має наступні переваги:

- Забезпечує незалежну перевірку цілісності файлів через підпис.
- Виключає можливість підробки змін у коді без схвалення відповідальної особи.
- Інтеграція з CI/CD забезпечує автоматизовану перевірку без ручного втручання.
- Прозорість та можливість аудиту змін через збереження метаданих.
- Може бути інтегрованим з SIEM та SOAR системами.

Також слід зазначити переваги використання даного методу для відповідності міжнародним стандартам з кібербезпеки.

Нижче наведено таблицю 2.2 з оглядом відповідності вдосконаленого методу стандарту ISO27001, а також рекомендаціям NIST та CIS Controls.

Таблиця 2.2 – Відповідність стандартам

Параметр	Відповідність ISO27001	Відповідність NIST	Відповідність CIS Controls
Цілісність даних	Забезпечується цілісність файлів за рахунок цифрового підпису (A.10.1.1)	Контроль хешів та цифрового підпису узгоджується з NIST SP 800-57 і SP 800-63	CIS Control 5 (Secure Configuration for Hardware and Software on Devices) – перевірка змін та забезпечення їхньої цілісності.
Автентичність даних	Цифрові підписи гарантують автентичність джерела даних (A.10.1.2)	Застосування асиметричних криптографічних ключів для підтвердження автентичності	CIS Control 13 (Data Protection) – забезпечення автентичності під час доступу до даних.
Контроль доступу	Обмеження доступу до ключів та процесу підпису регулюється політиками доступу (A.9.4.1)	NIST SP 800-53 AC-3 (Access Enforcement) – доступ обмежений ролями	CIS Control 4 (Controlled Use of Administrative Privileges) – захищений доступ до ключів.
Реагування на інциденти	Можливість виявлення спроб підробки змін і створення логів (A.16.1.1)	NIST Incident Response Guide (SP 800-61) – ідентифікація і сповіщення про інциденти	CIS Control 17 (Incident Response and Management) – сповіщення у разі виявлення загроз.
Автоматизація перевірок	Автоматизація описана у рекомендаціях ISO (A.12.6.1)	NIST SP 800-137 (Continuous Monitoring) – перевірки через автоматизовані процеси	CIS Control 18 (Application Software Security) – забезпечення автоматичних перевірок.
Стандарти передачі даних	Використання цифрових підписів відповідає принципам ISO (A.10.1.4)	Рекомендації NIST щодо безпечного обміну даними (SP 800-52, SP 800-77)	CIS Control 13 (Data Protection) – захист переданих даних.

Таким чином, впровадження вдосконаленого методу автентифікації на основі цифрового підпису в системах контролю версій значно підвищує рівень захисту та забезпечує ефективну перевірку цілісності та автентичності змін у коді. Цей підхід

дозволяє значно знизити ризики, пов’язані з несанкціонованим доступом, компрометацією ключів або токенів, підробкою змін у коді та соціальною інженерією. Крім того, вдосконалена система забезпечує прозорість процесів та можливість автоматизованого аудиту, що підвищує рівень довіри до змісту змін і дозволяє швидше реагувати на потенційні загрози.

Додатково, впроваджений метод автентифікації відповідає вимогам міжнародних стандартів кібербезпеки, таких як ISO27001, NIST та CIS Controls, що гарантує його відповідність сучасним вимогам щодо захисту даних та кібербезпеки. У сукупності ці переваги роблять систему не лише більш безпечною, а й більш ефективною для інтеграції в сучасні процеси розробки програмного забезпечення, що включають автоматизацію перевірок і забезпечення безпеки на кожному етапі.

2.4 Обґрунтування вибору криптографічних алгоритмів

Для забезпечення автентифікації та цілісності файлів у системах контролю версій важливим кроком є вибір надійних криптографічних алгоритмів, які гарантують безпеку та ефективність процесу підписування. У цьому підрозділі розглянемо основні вимоги до алгоритмів, обґрунтуємо їх вибір та дослідимо оптимальність використання кожного з них у розробленому алгоритмі.

Основними вимогами до криптографічних алгоритмів є:

1. Алгоритми повинні забезпечувати високу криптостійкість, щоб протистояти актуальним криптоаналітичним методам.
2. Алгоритми мають забезпечувати достатню швидкість обчислення для уникнення затримок у процесі автентифікації та роботі автоматичних систем.
3. Алгоритми повинні відповідати міжнародним стандартам криптографії та кібербезпеки загалом для забезпечення сумісності протягом довгого періоду часу.

На основі вказаних вимог було вирішено використовувати SHA-256 для хешування файлів та алгоритм цифрового підпису ECDSA.

Алгоритм SHA-256 є частиною сімейства алгоритмів SHA-2 і забезпечує високу стійкість до колізій. SHA-256 обрано через його відповідність стандартам та широкий рівень підтримки у більшості криптографічних бібліотек. Ключовими перевагами алгоритму є:

- Надійність проти колізій, оскільки SHA-256 генерує 256-бітний хеш, що робить його стійким до колізій і атак на вибіркові колізії, які критичні для збереження унікальності файлів;
- SHA-256 забезпечує високу швидкість хешування навіть для великих файлів, що мінімізує затримки при генерації хеш-значень для підписування.

Для створення цифрових підписів обрано алгоритм ECDSA, який базується на еліптичних кривих і є сучасним стандартом для криптографічного підписування. Основними перевагами використання ECDSA є:

- ECDSA дозволяє досягти високого рівня безпеки при значно меншій довжині ключів, що прискорює обчислення і знижує витрати на зберігання ключів.
- ECDSA забезпечує швидку генерацію та перевірку підпису, що є важливим для автоматизованих систем контролю версій, де підписування може виконуватися для великої кількості файлів.
- ECDSA затверджений у стандартах NIST і широко підтримується у багатьох програмних бібліотеках, що робить його оптимальним вибором для розробки сумісних і надійних систем.

Застосування алгоритмів SHA-256 для хешування та ECDSA для підписування файлів забезпечує надійний рівень захисту від несанкціонованих змін та підробок. Вибір цих алгоритмів обумовлений їх криптостійкістю, відповідністю міжнародним стандартам, а також високою ефективністю, що дозволяє використовувати їх у великих проєктах без шкоди для швидкості й продуктивності. Така система автентифікації та контролю версій сприяє надійному захисту від несанкціонованих модифікацій та полегшує контроль автентичності коду.

2.5 Обґрунтування вибору засобів програмування, системи контролю версій та серверної архітектури

Дана розробка вимагає вибору оптимальних засобів програмування, надійної системи контролю версій та серверної архітектури, яка забезпечить захищене зберігання криптографічних ключів і метаданих. У цьому підрозділі обґрунтовано вибір кожного з компонентів, зокрема, мови програмування, інструментів для контролю версій і серверної архітектури, для ефективного впровадження алгоритмів підписування та перевірки даних.

Для реалізації контролю версій обрано Git, а саме - GitHub, як основну систему версійного контролю, враховуючи такі переваги:

- GitHub є найбільш поширеною системою контролю версій, яка підтримує одночасну роботу декількох розробників, що дозволяє організувати контроль версій і відстежувати зміни в коді.
- GitHub легко інтегрується з інструментами автоматизованого тестування і розгортання, що дозволяє додати етапи перевірки підписів у даний процес [30].

Це дозволяє інтегрувати алгоритми цифрового підпису у процес контролю версій і забезпечує високий рівень захисту від несанкціонованих змін у кодовій базі.

Для забезпечення безпечного зберігання криптографічних ключів та метаданих у системі контролю версій обрано централізовану серверну архітектуру з використанням реляційної бази даних MySQL. Рішення обрано з огляду на такі критерії:

- Надійне зберігання та контроль доступу.
- Масштабованість та висока доступність.
- Шифрування на рівні бази даних.

MySQL забезпечує безпечне зберігання приватних і публічних ключів, а також метаданих підписів, пропонуючи розширене управління доступом через ролі та права користувачів, що обмежує доступ до ключів лише авторизованим

особам. Вказана СУБД підтримує кластерну архітектуру з реплікацією, що гарантує високу доступність та стійкість до відмов, що є критичним для масштабних систем контролю версій, також MySQL дозволяє шифрування даних на рівні зберігання, забезпечуючи додатковий захист приватних ключів навіть у разі компрометації сервера [32].

При виборі засобів програмування для реалізації функцій підписування, перевірки та зберігання ключів враховувалися такі критерії:

- Наявність бібліотек для роботи з криптографією.
- Продуктивність і надійність.
- Сумісність із базами даних.

На основі цих вимог обрано мову Python для реалізації системи вдосконалення автентифікації, оскільки Python забезпечує широкий вибір криптографічних бібліотек, таких як PyCryptodome та cryptography, які підтримують алгоритми SHA-256 та ECDSA, Python підтримує високопродуктивну взаємодію з базами даних через бібліотеки MySQL Connector або SQLAlchemy, що дозволяє безпечно зберігати та обробляти ключі. Також Python є популярним вибором для автоматизації процесів у системах контролю версій, що сприяє легкому впровадженню функцій цифрового підпису в середовище розробки [33].

2.6 Висновки до розділу

У даному розділі було розроблено алгоритм додаткової автентифікації на основі цифрового підпису та наведено обґрунтування вибору конкретних криптографічних алгоритмів для забезпечення цілісності та автентичності вихідного коду. Запропонований алгоритм забезпечує високий рівень захисту від несанкціонованих змін, що критично важливо для підтримки надійності систем контролю версій.

Вибір алгоритму хешування SHA-256 дозволяє уникнути колізій та зберегти унікальність даних, що підвищує стійкість до криптоаналітичних атак. Натомість використання алгоритму цифрового підпису ECDSA на основі еліптичних кривих

забезпечує високу швидкість роботи й низькі вимоги до обчислювальних ресурсів, що оптимально підходить для потреб масштабних систем версійного контролю. Інтеграція цих алгоритмів у процес розробки гарантує високу продуктивність, сумісність із сучасними стандартами безпеки та зручність використання у рамках автоматизованих систем.

Таким чином, розроблений алгоритм автентифікації не тільки відповідає вимогам сучасної кібербезпеки, а й забезпечує гнучкість і надійність при його впровадженні в масштабних програмних проєктах. Це рішення робить можливим контроль за автентичністю та цілісністю коду, що є основою для підтримання високої якості та безпеки програмного забезпечення.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ВДОСКОНАЛЕНОГО МЕТОДУ АВТЕНТИФІКАЦІЇ В СИСТЕМАХ КОНТРОЛЮ ВЕРСІЙ ВИХІДНОГО КОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ ЦИФРОВОГО ПІДПИСУВАННЯ

3.1 Розробка клієнтської частини додатка

В даному розділі опишемо основні фрагменти коду, який використовується для вдосконалення методу авторизації у системах контролю версій вихідного коду з використанням цифрового підписування зі сторони клієнта.

Першою функцією даного додатку є генерація пари ключів - приватного та публічного для подальшого підпису файлів.

Для виконання даного функціоналу імпортуються необхідні бібліотеки:

```
import os
import hashlib
from ecdsa import SigningKey, NIST384p
```

Отримуємо необхідні дані для генерації ключів, а саме - ім’я та прізвище та шлях для збереження даних:

```
first_name = input("Введіть ім'я: ")
last_name = input("Введіть прізвище: ")
save_path = input("Введіть шлях для збереження ключів (натисніть Enter для поточної директорії): ")
if not save_path:
save_path = os.getcwd()
```

Після чого викликається функція генерації ключів:

```
def generate_keys():
    ECDSA
    private_key = SigningKey.generate(curve=NIST384p)
```

```
public_key = private_key.verifying_key
return private_key, public_key
```

Після генерації ключів для них створюється назва з імені та прізвища та виконується збереження за вказаним шляхом:

```
private_key_filename = f"{first_name}_{last_name}_private_key.pem"
public_key_filename = f"{first_name}_{last_name}_public_key.pem"
private_key_path = os.path.join(save_path, private_key_filename)
public_key_path = os.path.join(save_path, public_key_filename)
save_key_to_file(private_key, private_key_path)
save_key_to_file(public_key, public_key_path)
```

Наступним кроком є реалізація коду для підпису файлів проєкту. В даному коді імпортуються наступні бібліотеки:

```
import os
import hashlib
import requests
from ecdsa import SigningKey, NIST384p
```

Спочатку в консолі виконується збір інформації для роботи, а саме - запитується шлях до приватного ключа та директорії проєкту, при введені даних значень відбувається їх перевірка на коректність:

```
private_key_path = input("Введіть шлях до приватного ключа: ")
if not os.path.exists(private_key_path):
print("Невірний шлях до приватного ключа.")
return
directory = input("Введіть шлях до директорії з файлами (натисніть Enter для поточної): ")
if not directory:
```

```
directory = os.getcwd()
```

Після отримання вхідних даних та їх верифікації виконується обрахунок хеш суми та підпису кожного файлу, за виключенням службових занчень системи контролю версів вихідного коду програмного забезпечення та відправка значень разом з ім’ям файлу на сервер сховища:

```
for dirpath, dirnames, filenames in os.walk(directory):
    dirnames[:] = [d for d in dirnames if d != '.git']
     for filename in filenames:
    if '.git' in dirpath:
        continue
    file_path = os.path.join(dirpath, filename)
    if os.path.isfile(file_path):
        file_hash = generate_file_hash(file_path)
        signature = sign_data(private_key_path, file_hash)
        print(f"Підписуємо та відправляємо файл: {filename}")
    response = requests.post('http://X.X.X.X:5000/upload', json={
        'filename': filename,
        'file_hash': file_hash,
        'signature': signature
    })
    if response.status_code == 201 or response.status_code == 200:
        print(f"Файл '{filename}' успішно відправлено на сервер.")
    else:
        print(f"Помилка відправки файлу '{filename}': {response.json().get('error', 'Unknown error')}")
```

Для створення хеш-суми використовується функція generate_file_hash, в якому передається шлях до поточного файлу, над яким виконуються роботи:

```
def generate_file_hash(file_path):
    sha256_hash = hashlib.sha256()
    with open(file_path, 'rb') as file:
    for byte_block in iter(lambda: file.read(4096), b""):
    sha256_hash.update(byte_block)
    return sha256_hash.hexdigest()
```

Для підпису з використанням приватного ключа в наступну функцію передається шлях до файлу-ключа та хеш сума:

```
def sign_data(private_key_path, data):
    with open(private_key_path, 'rb') as key_file:
    private_key = SigningKey.from_pem(key_file.read())
    signature = private_key.sign(data.encode(), hashfunc=hashlib.sha256)
    return signature.hex()
```

Варто зазначити, що даний функціонал виконується з високим рівнем перевірки коректності введенних значень, а також відповідей сервера.

Останнім функціоналом, який виконується зі сторони клієнта є верифікація файлів, яку можливо виконати в будь-який момент.

В його роботі задіяно наступні бібліотеки мови Python:

```
import os
import hashlib
import requests
import sys
```

Для цього при запуску програми запитується шлях до директорії проєкту та запускається рекурсивна перевірка для кожного файлу в даній директорії, у разі невідповідності хоча б одного файлу - повідомляється про порушення безпеки:

```
def main():
```

```
error_occurred = False
directory = input("Введіть шлях до директорії для верифікації (натисніть Enter для поточної): ")
if not directory:
directory = os.getcwd()
for dirpath, dirnames, filenames in os.walk(directory):
dirnames[:] = [d for d in dirnames if d != '.git']
for filename in filenames:
if '.git' in dirpath:
    continue
file_path = os.path.join(dirpath, filename)
if os.path.isfile(file_path):
    print(f"Перевірка файлу: {filename}")
    if not verify_file_on_server(filename, file_path):
    error_occurred = True
if error_occurred:
print("Є несанкціоновані зміни!")
sys.exit(1)
print("Всі файли були верифіковані успішно.")
```

В даному коді використовується дві функції - generate_file_hash з аналогічним функціоналом, який було описано вище, дана функція обраховує поточний хеш файлів. Друга функція надсилає дані на сервер сховище та перевіряє статус відповіді, що далі передається в головну функцію для виводу повідомлень аудитору.

3.2 Розробка серверної частини додатка

Для розробки додатку на сервері сховищі було обрано фреймворк мови Python Flask. Для реалізації функціоналу при запуску програми імпортуються наступні бібліотеки:

```
from flask import Flask, request, jsonify
import mysql.connector
from ecdsa import VerifyingKey, BadSignatureError
import hashlib
import os
```

Також налаштовується підключення до бази даних MySQL з використанням змінних оточення з описом відповідної функції:

```
db_config = {
    'host': os.getenv('DB_HOST'),
    'user': os.getenv('DB_USER'),
    'password': os.getenv('DB_PASSWORD'),
    'database': os.getenv('DB_NAME')
}
def connect_db():
    return mysql.connector.connect(**db_config)
```

На даному етапі додаток можна умовно розділити на дві функції, які розмежовуються відповідними шляхами в URL посиланні веб-сервера Flask, а саме upload_file - для завантаження підписаних файлів та verify_file - для перевірки незмінності файлів проєкту.

При завантаженні файлів по API отримуються наступні дані - ім’я файлу, його хеш та підпис:

```
data = request.get_json()
filename = data['filename']
file_hash = data['file_hash']
signature = data['signature']
```

Наступним кроком отримується публічний ключ з бази даних, при цьому виконується перевірка на коректність даних:

```
  conn = connect_db()
cursor = conn.cursor()
cursor.execute("SELECT public_key FROM trusted_public_keys LIMIT 1")
result = cursor.fetchone()
if result is None:
return jsonify({'error': 'Public key not found'}), 404
public_key = result[0]
```

Після отримання даних виконується перевірка відповідності цифрового підпису до публічного ключа, збереженого в базі даних:

```
if not verify_signature(public_key, signature, file_hash):
    return jsonify({'error': 'Invalid signature'}), 400
    cursor.execute("SELECT * FROM files WHERE filename = %s", (filename,))
    existing_file = cursor.fetchone()
```

Для цього використовується функція verify_signature:

```
def verify_signature(public_key, signature, file_hash):
    try:
    vk = VerifyingKey.from_pem(public_key)
    vk.verify(bytes.fromhex(signature), file_hash.encode(), hashfunc=hashlib.sha256)
    return True
```

```
except BadSignatureError:
return False
```

Таким чином запобігаються випадки, коли зловмисник намагається підписати файли з нелегітимним ключем.

Якщо ключ дійсний, завантажуються отримані файли, при цьому виконується перевірка - файл існує в базі чи ні, якщо файл новий - створюється новий запис, якщо існує - оновлюється поточний, що дозволяє знизити витрати на ресурси сервера:

```
if existing_file:
    update_query = """
    UPDATE files
    SET digital_signature = %s, public_key = %s, file_hash = %s
    WHERE filename = %s
    """
    cursor.execute(update_query, (signature, public_key, file_hash, filename))
    conn.commit()
    return jsonify({'message': 'File metadata updated successfully'}), 200
    else:
       insert_query = """
    INSERT INTO files (filename, file_hash, digital_signature, public_key)
    VALUES (%s, %s, %s, %s)
    """
    cursor.execute(insert_query, (filename, file_hash, signature, public_key))
    conn.commit()
    return jsonify({'message': 'File metadata saved successfully'}), 201
    except Exception as e:
    return jsonify({'error': str(e)}), 500
```

Для верифікації підпису виконуються аналогічні шляхи ініціалізації, а саме - отримання значень від клієнт та підключення до бази даних, після чого завантажується звідти список файлів з їх характеристиками та виконується звірка на наявність отриманих файлів у базі даних, у випадку відсутності файлу - про це повідомляється та в таблицю incidents записується інформація про виникнення даної ситуації, з збереженням назви файлу, типом невідповідності, ip-адресою хоста, на якому виконано перевірку та часом інциденту: @app.route('/verify', methods=['POST'])

```
def verify_file():
    try:
    data = request.json
    filename = data['filename']
    file_path = data.get('file_path')
    current_hash = data.get('file_hash')
    conn = connect_db()
    cursor = conn.cursor(dictionary=True)
    select_query = "SELECT * FROM files WHERE filename = %s"
    cursor.execute(select_query, (filename,))
    file_record = cursor.fetchone()
    if not file_record:
    ip_address = request.remote_addr
    cursor.execute(
        "INSERT INTO incidents (filename, issue, ip_address) VALUES (%s, %s, %s)",
        (filename, 'File metadata not found', ip_address)
    )
    conn.commit()
    return jsonify({'error': 'File metadata not found in the database'}), 404
```

Наступним етапом перевірки є звірка підпису та хеш суми отриманого файлу з тими значеннями, які збережені в базі даних:

```
if current_hash != file_record['file_hash']:
ip_address = request.remote_addr
cursor.execute(
    "INSERT INTO incidents (filename, issue, ip_address) VALUES (%s, %s, %s)",
    (filename, 'File hash mismatch', ip_address)
)
conn.commit()
return jsonify({'error': 'File hash mismatch'}), 400


if not verify_signature(file_record['public_key'], file_record['digital_signature'], file_record['file_hash']):
ip_address = request.remote_addr
cursor.execute(
    "INSERT INTO incidents (filename, issue, ip_address) VALUES (%s, %s, %s)",
    (filename, 'Signature verification failed', ip_address)
)
conn.commit()
return jsonify({'error': 'Signature verification failed'}), 400
return jsonify({'message': 'File verification successful'}), 200
except Exception as e:
return jsonify({'error': str(e)}), 500
```

Варто зазаначити, що в даних випадках також записуються аналогічні дані про інцидент в базу даних для можливості інтеграції з SIEM та SOAR системами.

Для перевірки підпису викликається функція verify_signature, яка має наступний вигляд:

```
def verify_signature(public_key, signature, file_hash):
    try:
    vk = VerifyingKey.from_pem(public_key)
    vk.verify(bytes.fromhex(signature), file_hash.encode(), hashfunc=hashlib.sha256)
    return True
    except BadSignatureError:
    return False
```

Окремо слід виділити функціонал автоматизованої перевірки наявності несанкціонованих змін в системі контролю версій вихідного коду програмного забезпечення. В даній роботі використовувався функціонал GitHub Actions. В файлі main.yml вказано workflow, який виконує дані дії:
name: Demonstration of improved protection

```
on:
  workflow_dispatch:
jobs:
  signature_verification:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - name: Check signature
      run: echo . | python verify.py
  other_actions:
    needs: signature_verification
    runs-on: ubuntu-latest
```

steps:

- run: echo "other actions are performed"

Дані налаштування демонструють принцип роботи перевірки - під час автоматичного розгортання виконується отримання поточних файлів репозиторію, викликається функція верифікації даних та у випадку невідповідності - інші кроки не виконуються.

3.3 Тестування програмного продукту

Для генерації ключів регулюючій особі необхідно запустити файл keygen.py, після чого в інтерактивній консолі надати ім’я та прізвище, а також вказати шлях для збереження файлів ключів. Після введення даних пару ключів буде успішно згенеровано (рис.3.1).

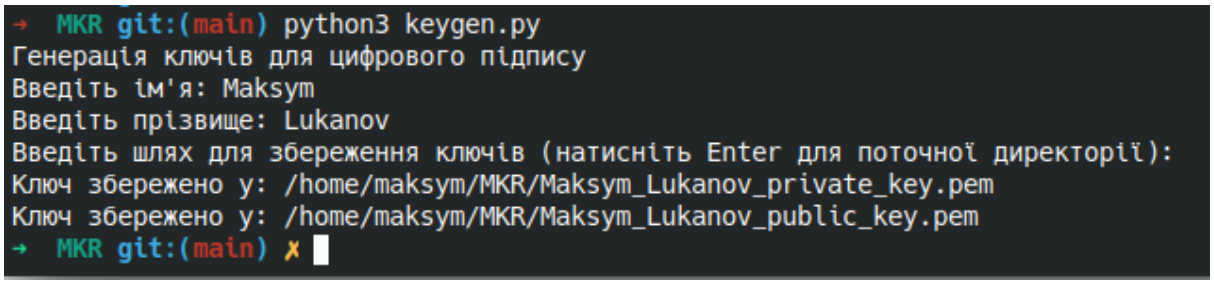


Рисунок 3.1 – Генерація ключів для підпису

Для підпису файлів необхідно запустити файл signing.py, після чого вказати шлях до приватного ключа та директорії проекту. Після введення даних файли будуть підписано та надіслано на сервер (рис. 3.2).

```
Введіть шлях до приватного ключа: ../keys_for_MKR/Maksym_Lukanov_private_key.pem
Введіть шлях до директорії з файлами (натисніть Enter для поточної):
Підписуємо та відправляємо файл: keygen.py
Файл 'keygen.py' успішно відправлено на сервер.
Підписуємо та відправляємо файл: .gitignore
Файл '.gitignore' успішно відправлено на сервер.
Підписуємо та відправляємо файл: verify.py
Файл 'verify.py' успішно відправлено на сервер.
Підписуємо та відправляємо файл: signing.py
Файл 'signing.py' успішно відправлено на сервер.
Підписуємо та відправляємо файл: .env
Файл '.env' успішно відправлено на сервер.
Підписуємо та відправляємо файл: server.py
Файл 'server.py' успішно відправлено на сервер.
Підписуємо та відправляємо файл: docker-compose.yml
Файл 'docker-compose.yml' успішно відправлено на сервер.
Підписуємо та відправляємо файл: requirements.txt
Файл 'requirements.txt' успішно відправлено на сервер.
Підписуємо та відправляємо файл: Dockerfile
Файл 'Dockerfile' успішно відправлено на сервер.
→  MKR git:(main)
```

Рисунок 3.2 – Підпис файлів проекту

Верифікація файлів ініціалізується запуском verify.py. Після запуску програмного застосунку необхідно вказати шлях до файлів проекту (рис. 3.3). Варто зазначити, що для перевірки наявність ключів в користувача не вимагається, що дозволяє виконувати перевірку в різному середовищі, в тому числі - автоматизованому в workflow систем контролю версій вихідного коду програмного забезпечення або на кінцевих точках.

```
→  MKR git:(main) python3 verify.py
Введіть шлях до директорії для верифікації (натисніть Enter для поточної):
Перевірка файлу: keygen.py
Верифікація файлу 'keygen.py' пройшла успішно.
Перевірка файлу: .gitignore
Верифікація файлу '.gitignore' пройшла успішно.
Перевірка файлу: verify.py
Верифікація файлу 'verify.py' пройшла успішно.
Перевірка файлу: signing.py
Верифікація файлу 'signing.py' пройшла успішно.
Перевірка файлу: .env
Верифікація файлу '.env' пройшла успішно.
Перевірка файлу: server.py
Верифікація файлу 'server.py' пройшла успішно.
Перевірка файлу: docker-compose.yml
Верифікація файлу 'docker-compose.yml' пройшла успішно.
Перевірка файлу: requirements.txt
Верифікація файлу 'requirements.txt' пройшла успішно.
Перевірка файлу: Dockerfile
Верифікація файлу 'Dockerfile' пройшла успішно.
Всі файли були верифіковані успішно.
→  MKR git:(main)
```

Рисунок 3.3 – Верифікація файлів, запущена користувачем

Під час використання перевірки в автоматизованій системі GitHub Actions перевірка запускається при старті workflow, в якому першим кроком є перевірка незмінності файлів (рис. 3.4).

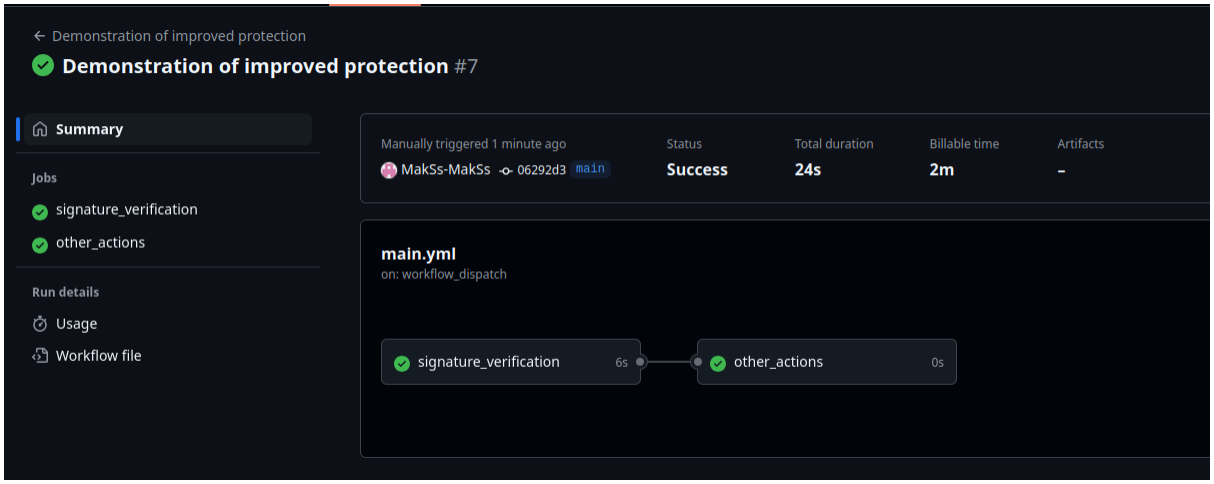


Рисунок 3.4 – Покроковість роботи перевірки

Таким чином виникнення помилки на першому кроці зупиняє подальше виконання автоматизованої системи.

Також адміністратор або DevOps спеціаліст матиме змогу переглянути деталі перевірки (рис. 3.5)

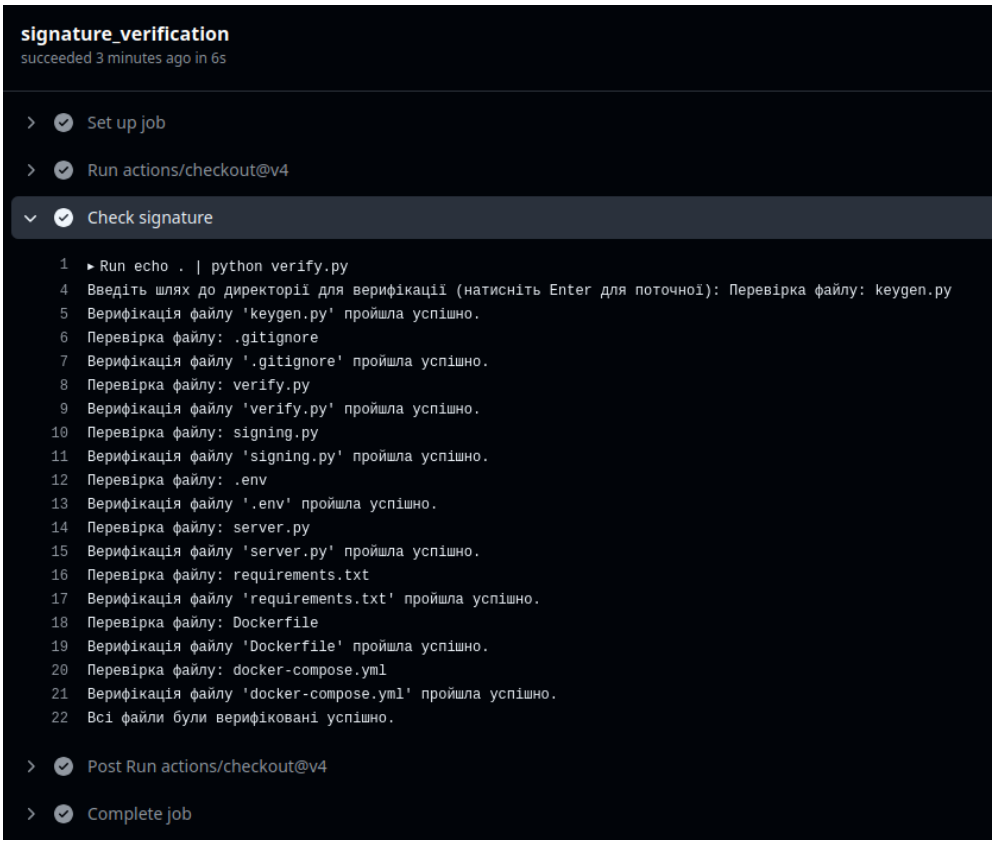
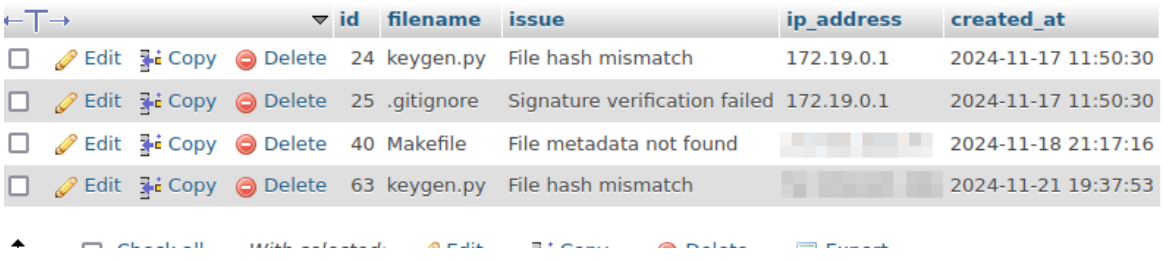


Рисунок 3.5 – Деталі перевірки в автоматизованій системі

Окремим функціоналом можна виділити графічний інтерфейс СУБД MySQL, який використовуватиметься для керування та моніторингу програмного додатку, а також отриманням інформації про інциденти в таблиці incidents, яка містить інформацію про кожну ситуацію невідповідності файлів під час виконання верифікації, а саме - файл, який не відповідає, час фіксування інциденту, ip-адреса з якої виконувались запити на верифікацію та тип невідповідності, це може бути невідповідність хешу, підпису або відсутність інформації про даний файл (рис. 3.6)

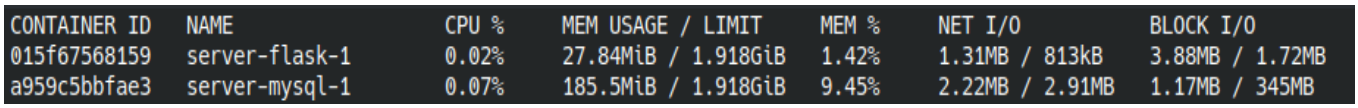


	id	filename	issue	ip_address	created_at
Edit Copy Delete	24	keygen.py	File hash mismatch	172.19.0.1	2024-11-17 11:50:30
Edit Copy Delete	25	.gitignore	Signature verification failed	172.19.0.1	2024-11-17 11:50:30
Edit Copy Delete	40	Makefile	File metadata not found	[redacted]	2024-11-18 21:17:16
Edit Copy Delete	63	keygen.py	File hash mismatch	[redacted]	2024-11-21 19:37:53

Рисунок 3.6 – Таблиця incidents

Таким чином цей функціонал дозволить зберігати велику кількість записів про інциденти, а також налаштувати інтеграцію з SIEM та SOAR системами, які використовується в підприємстві.

Також проведено тестування споживання ресурсів серверної частини додатку (рис. 3.7).



CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O
015f67568159	server-flask-1	0.02%	27.84MiB / 1.918GiB	1.42%	1.31MB / 813kB	3.88MB / 1.72MB
a959c5bbfae3	server-mysql-1	0.07%	185.5MiB / 1.918GiB	9.45%	2.22MB / 2.91MB	1.17MB / 345MB

Рисунок 3.7 – Використання ресурсів серверною частиною

Загальне споживання ресурсів є наступним:

1. Використання ресурсів процесора становить в сумі 0.09% ресурсу ядра. Перевірка відбувалась на процесорі з частотою 2800MHz в режимі реального часу.
2. Використання ресурсів оперативної пам’яті в сумі становить 213,34 Mib в режимі реального часу.
3. Споживання трафіку є невеликим – 3.51MB вхідного та 3,53MB вихідного трафіку протягом 15 днів.
4. Використання потужності дискового накопичувача також є незначним – 5,05MB запису та 346,72MB протягом 15 днів роботи.

Окремо слід вказати структуру бази даних, яка завдяки простій реалізації споживає мінімально ресурсів (рис. 3.8).

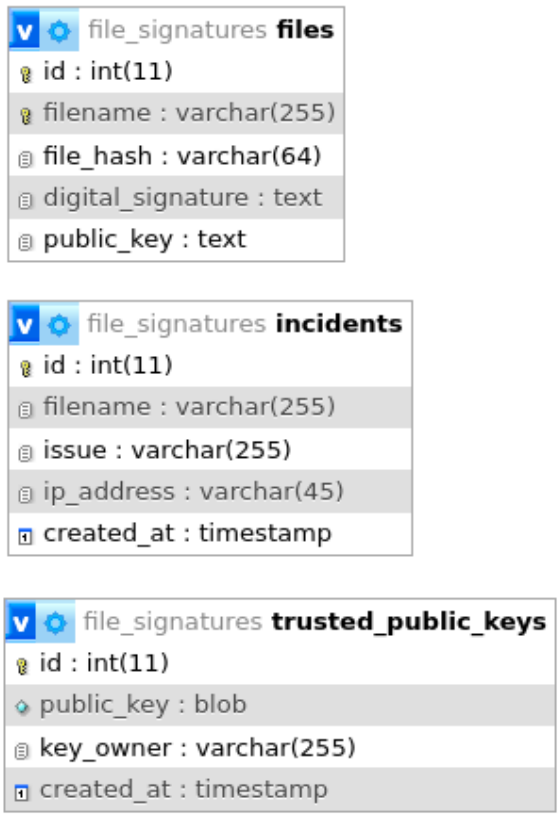


Рисунок 3.8 – Структура бази даних

Згідно даної структури існує три таблиці, в яких вказано мінімум необхідної інформації та відсутні зв’язки, оскільки дані використовуються виключно в форматі збереження та не вимагають складної архітектури.

Для перевірки стійкості до атак підпишемо файли проєкту згідно інструкції, описаною в даному розділі.

Після підпису виконаємо перевірку (рис. 3.9).

```
→  MKR git:(main) python3 verify.py
Введіть шлях до директорії для верифікації (натисніть Enter для поточної):
Перевірка файлу: keygen.py
Верифікація файлу 'keygen.py' пройшла успішно.
Перевірка файлу: .gitignore
Верифікація файлу '.gitignore' пройшла успішно.
Перевірка файлу: verify.py
Верифікація файлу 'verify.py' пройшла успішно.
Перевірка файлу: signing.py
Верифікація файлу 'signing.py' пройшла успішно.
Перевірка файлу: .env
Верифікація файлу '.env' пройшла успішно.
Перевірка файлу: server.py
Верифікація файлу 'server.py' пройшла успішно.
Перевірка файлу: docker-compose.yml
Верифікація файлу 'docker-compose.yml' пройшла успішно.
Перевірка файлу: requirements.txt
Верифікація файлу 'requirements.txt' пройшла успішно.
Перевірка файлу: Dockerfile
Верифікація файлу 'Dockerfile' пройшла успішно.
Всі файли були верифіковані успішно.
→  MKR git:(main)
```

Рисунок 3.9 – результати перевірки файлів на початку тестування

Оскільки підписані файли не були змінені, верифікація пройшла успішно.

Тепер внесемо зміни у файл keygen.py, а також створимо новий файл test.file, який відсутній в підписаній версії та виконаємо повторну перевірку (3.10).

```
→  MKR git:(main) echo  "#test" >> keygen.py
→  MKR git:(main) ✗ touch test.file
→  MKR git:(main) ✗ python3 verify.py
Введіть шлях до директорії для верифікації (натисніть Enter для поточної):
Перевірка файлу: keygen.py
Помилка верифікації файлу 'keygen.py': File hash mismatch
Перевірка файлу: .gitignore
Верифікація файлу '.gitignore' пройшла успішно.
Перевірка файлу: test.file
Помилка верифікації файлу 'test.file': File metadata not found in the database
Перевірка файлу: verify.py
Верифікація файлу 'verify.py' пройшла успішно.
Перевірка файлу: signing.py
Верифікація файлу 'signing.py' пройшла успішно.
Перевірка файлу: .env
Верифікація файлу '.env' пройшла успішно.
Перевірка файлу: server.py
Верифікація файлу 'server.py' пройшла успішно.
Перевірка файлу: docker-compose.yml
Верифікація файлу 'docker-compose.yml' пройшла успішно.
Перевірка файлу: requirements.txt
Верифікація файлу 'requirements.txt' пройшла успішно.
Перевірка файлу: Dockerfile
Верифікація файлу 'Dockerfile' пройшла успішно.
Є несанкціоновані зміни!
→  MKR git:(main) ✗
```

Рисунок 3.10 – верифікація файлів при несанкціонованій зміні.

В результаті перевірки встановлено, що вказані файли були змінено та верифікацію не пройдено.

Для симуляції ситуації, коли підпис не відповідає файлу з аналогічним хешем, в базі даних змінимо інформацію про підпис файлу keygen.py та виконаємо верифікацію повторно (рис 3.11).

```
→  MKR git:(main) ✗ python3 verify.py
Введіть шлях до директорії для верифікації (натисніть Enter для поточної):
Перевірка файлу: keygen.py
Помилка верифікації файлу 'keygen.py': Signature verification failed
Перевірка файлу: .gitignore
Верифікація файлу '.gitignore' пройшла успішно.
Перевірка файлу: test.file
```

Рисунок 3.11 – верифікації при недійсному підписі

Згідно результатів верифікації, знайдено що підпис не відповідає публічному ключу, який збережено в базі даних, отже – файл було змінено.

Також інформація про інциденти збережено в базі даних для можливості інтеграції з SIEM та SOAR системами (рис. 3.12)

id	filename	issue	ip_address	created_at ▾ 1
71	keygen.py	Signature verification failed	[redacted]	2024-12-[redacted]
68	keygen.py	File hash mismatch	[redacted]	2024-12-[redacted]
69	test.file	File metadata not found	[redacted]	2024-12-[redacted]

Рисунок 3.12 – інформація про інциденти, які були створені під час тестування

На основі проведених тестів можна зробити висновки, що застосунок не використовує великої кількості ресурсів та надійно захищає від несанкціонованих змін у файлах.

3.4 Висновки до розділу

У даному розділі було здійснено розробку клієнтської та серверної частини програмного додатку для вдосконалення методу авторизації у системах контролю версій вихідного коду з використанням цифрового підпису.

У рамках клієнтської частини реалізовано функціонал генерації пари ключів, підпису файлів проекту, верифікації файлів, а також автоматизовану перевірку незмінності файлів із застосуванням GitHub Actions. Було імпортовано необхідні бібліотеки Python, розроблено алгоритми для хешування файлів, цифрового підпису та відправлення даних на сервер. Описані функції забезпечують високу надійність та перевірку коректності введених даних.

На серверній стороні використано фреймворк Flask для створення API для завантаження та верифікації файлів. Реалізовано інтеграцію з базою даних MySQL для зберігання публічних ключів, хеш-сум та підписів файлів. Також впроваджено механізми реєстрації інцидентів невідповідності файлів у таблиці incidents, що забезпечує можливість інтеграції з SIEM та SOAR системами.

Розроблений функціонал дозволяє ефективно виявляти несанкціоновані зміни у проектах, що підтверджено тестуванням, підвищує рівень безпеки систем контролю версій та створює умови для автоматизованого моніторингу, а також є ощадним до серверних ресурсів.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінювання потенціалу розробки програмного забезпечення

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності [34].

Результатом магістерської кваліфікаційної роботи є розробка програмного засобу для реалізації вдосконаленого методу автентифікації в системах контролю версій вихідного коду програмного забезпечення з використанням цифрового підписування.

Для проведення технологічного аудиту залучено трьох незалежних експертів. У межах даної роботи такими експертами є викладачі кафедри МБІС: Грицак А. В. (доц., викл. каф. МБІС ВНТУ); Салієва О.В. (д.ф., викл. каф. МБІС ВНТУ); Карпінець В. В. (к.т.н., доцент каф. МБІС ВНТУ).

Оцінювання комерційного потенціалу буде здійснено за критеріями, що наведені в таблиці 4.1

Таблиця 4.1 – Критерії оцінювання комерційного потенціалу розробки

Критерії оцінювання та 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 роботоздатність 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 аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
5	Експлуатацій ні витрати значно вищі, ніж в аналогів	Експлуатацій ні витрати дещо вищі, ніж в аналогів	Експлуатацій ні витрати на рівні експл. витрат аналогів	Експлуатацій ні 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 ринку	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 навч. 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 використовую ться у військовопромисло вом у 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 к-ті дозвільних документів на вир-во 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 комерційного потенціалу експертами розробки зведено в таблицю 4.2.

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки.

Критерії	Прізвище, ініціали, посада експерта		
	1 – Грицак А.В.	2 – Салієва О.В	3 – Карпінець В.В.
1	4	4	4
Ринкові переваги (недоліки):			
2	3	3	4
3	4	4	4
4	4	4	4
5	4	3	3
Ринкові перспективи			
6	4	4	3
7	4	3	4
Практична здійсненність			
8	4	3	4
9	3	4	4
10	4	4	3
11	3	4	4
12			
Сума балів	СБ1 = 41	СБ2 = 40	СБ3 = 41
Середньоарифметична сума балів СБ	СБ = 41		

За даними таблиці 4.2 можна зробити висновок, щодо рівня комерційного потенціалу розробки. Врахуємо на результат й порівняємо його з рівнями комерційного потенціалу розробки, що представлено в таблиці 4.3.

Таблиця 4.3 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 41 бал, тому згідно таблиці 4.2 вважається, що рівень комерційного потенціалу проведених досліджень є високий.

4.2 Прогнозування витрат на виконання наукової роботи та впровадження її результатів

Прогнозування витрат на виконання науково-дослідної роботи складається з таких етапів:

1-й етап: розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи;

2-й етап: розрахунок загальних витрат на виконання даної роботи;

3-й етап: прогнозування загальних витрат на виконання та впровадження результатів даної роботи.

Виконаємо розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи, за такими статтями та формулами, приймаючи до уваги те, що для розробки інформаційної технології було залучено одного розробника програмного забезпечення.

1. Основна заробітна плата $З_о$:

$$З_о = \frac{М}{Т_р} * t \quad \text{(грн)}, \qquad (4.1)$$

де М – місячний посадовий оклад – 32 000 грн. ;

$Т_р$ – число робочих днів в місяці; приблизно $Т_р$ = 22 днів;

t – число робочих днів для всієї роботи – 44 днів.

Таким чином:

$$З_о = \frac{32\,000}{22} * 44 = 64\,000 \text{ (грн.)}$$

Таблиця 4.4 – Витрати по заробітній платі

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату
Розробник	32 000	1 454,5	44	64 000
Всього				64 000

2. Додаткова заробітна плата $З_д$ працівників розраховується як 12% від основної заробітної плати :

$$З_д = 0{,}12 * 64\,000 = 7\,680 \text{ (грн.)} – \text{для розробника}$$

3. Нарахування на заробітну плату $Н_{зп}$ розробника становить:

$$Н_{зп} = (З_о + З_д) * \frac{\beta}{100}, \qquad (4.2)$$

де $З_о$ – основна заробітна плата розробника;

$З_д$ – додаткова заробітна плата розробника;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування – 22%.

$$Н_{зп} = (64\,000 + 7\,680) * 0{,}22 = 15\,770$$

4. Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час виконання даного етапу роботи розраховуємо за формулою:

$$А = \frac{Ц * Т}{12 * Т_в} \qquad (4.3)$$

де Ц – загальна балансова вартість обладнання, приміщення тощо, грн.;

Т – фактична тривалість використання, міс;

$Т_в$ – термін використання обладнання, приміщень тощо, роки. Розробка програмного забезпечення ведеться орієнтовно 2 місяці.

Розрахунки зведено в таблиці 4.6:

Таблиця 4.5 – Амортизаційні відрахування

Найменування	Балансова вартість (грн.)	Термін використання (років)	Фактична тривалість вня, (міс.)	Величина ам.. відрахувань, (грн.)
Комп'ютер	22 000	4	2	917
Монітор	10 000	2	2	833
			Всього	1 750

5. Витрати на комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$А = \sum_{i=1}^{n} Н_i * Ц_i * К_i \qquad (4.4)$$

де $Н_i$ – кількість комплектуючих і-го виду, шт.;

$Ц_i$– ціна комплектуючих і-го виду, грн.;

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

n – кількість видів комплектуючих.

Таблиця 4.6 – Витрати на комплектуючі

Найменування комплектувальних	Кількість	Ціна за штуку, грн.	Сума, грн.	Примітка
Клавіатура	1	1 500	1 500	
Комп'ютерна мишка	1	1 000	1 000	
		Всього:	$К_i$=1,2	2 500 грн.

6. Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} * t_i * Ц_е * К_{впі}}{\mu_i} \quad (4.5)$$

де W – установлена потужність обладнання – 0,8 кВт;

t – тривалість роботи обладнання на етапі дослідження, год;

Ц – вартість 1 кВт-години електроенергії, грн; (станом на 01.10.2024 для підприємців вартість 4,32 грн./кВт-год)

К – коефіцієнт, що враховує використання потужності, К< 1;

μ_i – коефіцієнт корисної дії обладнання, $\mu_i < 1$.

$$В_е = \frac{0{,}8 * 352 * 4{,}32 * 0{,}5}{0{,}8} = 760 \text{ (грн.)}$$

7. Інші витрати $В_{ін}$ охоплюють:

− витрати на управління організацією;

− оплату службових відряджень;

− витрати на утримання, ремонт та експлуатацію основних засобів;

− витрати на опалення, освітлення, водопостачання, охорону праці тощо.

Інші витрати $В_{ін}$ можна прийняти як 100% від суми основної заробітної плати розробника:

$$В_{ін} = 64\,000 * 1 = 64\,000 \text{ (грн.)}$$

8. Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи – В.

$$В = 64\,000 + 7\,680 + 15\,770 + 1\,750 + 2\,500 + 760 + 64\,000 = 156\,460$$

9. Проведемо прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи. Прогнозування здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta}, \text{ грн.} \quad (4.6)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Так, якщо розробка знаходиться:

− на стадії науково-дослідних робіт, то $\beta \approx 0{,}1$;

− на стадії технічного проектування, то $\beta \approx 0{,}2$;

− на стадії розробки конструкторської документації, то $\beta \approx 0{,}3$;

− на стадії розробки технологій, то $\beta \approx 0{,}4$;

− на стадії розробки дослідного зразка, то $\beta \approx 0{,}5$;

− на стадії розробки промислового зразка, $\beta \approx 0{,}7$;

− на стадії впровадження, то $\beta \approx 0{,}9$.

$В_{заг}$ – загальна вартість всієї наукової роботи.

$$ЗВ = \frac{156\,460}{0{,}9} = 173\,844 \text{ (грн.)}$$

Отже, прогноз загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи складає 173 844 (грн.)

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі проведемо кількісне прогнозування, яку вигоду можна отримати у майбутньому від впровадження результатів виконаної наукової роботи.

В умовах ринку узагальнюючим позитивним результатом, що його отримує підприємство від впровадження результатів тієї чи іншої розробки, є збільшення чистого прибутку підприємства. Зростання чистого прибутку можна оцінити у теперішній вартості грошей.

Зростання чистого прибутку забезпечить інвестору надходження додаткових коштів, які дозволять покращити фінансові результати діяльності. Виконання даної наукової роботи та впровадження її результатів складає приблизно 1 рік. Позитивні результати від впровадження розробки очікуються вже в перші місяці після впровадження.

Проведемо детальне прогнозування позитивних результатів та кількісне їх оцінювання по роках.

Обчислимо збільшення чистого прибутку підприємства $\Delta П_і$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_і = \sum_1^n (\Delta Ц_о * N * Ц_о * \Delta N)_i * \lambda * \rho * \left(1 - \frac{\upsilon}{100}\right) \quad (4.7)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_о$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

ρ – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

υ– ставка податку на прибуток. У 2024 році – 18%.

Припустимо, що внаслідок впровадження результатів наукової розробки чистий прибуток підприємства збільшиться на 1200 грн., а кількість одиниць (наданих прав на користування сервісом) реалізованої послуги збільшиться:

− протягом першого року – на 90 од.,

− протягом другого року – ще на 70 од.,

− протягом третього року – ще на 50 од.

Орієнтовно: реалізація продукції до впровадження результатів наукової розробки складала 1 шт., а прибуток, що його отримувало підприємство на одиницю продукції до впровадження результатів наукової розробки – 20 000грн.

Потрібно спрогнозувати збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства $\Delta П_1$ протягом першого року складе:

$$\Delta П_1 = (1\,200 * 1 + (20\,000 + 1\,200) * 90) * 0{,}833 * 0{,}25 * \left(1 + \frac{18}{100}\right) =$$

469 157 (грн.)

Обчислимо збільшення чистого прибутку підприємства $\Delta П_2$ протягом другого року:

$$\Delta П_2 = (1\,200 * 1 + (20\,000 + 1\,200) * 70) * 0{,}833 * 0{,}25 * \left(1 + \frac{18}{100}\right) =$$

364 966 (грн.)

Збільшення чистого прибутку підприємства $\Delta П_з$ протягом третього року становитиме:

$$\Delta П_3 = (1\,200 * 1 + (20\,000 + 1\,200) * 50) * 0{,}833 * 0{,}25 * \left(1 + \frac{18}{100}\right) =$$

$260\,774$ (грн.)

Отже, розрахунки показують, що відповідно прогнозуванню комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку підприємства.

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Основними показниками, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \qquad (4.8)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{інв}$ = 2…5).

$$PV = 2 \cdot 173\,844 = 347\,688 \ \ (\text{грн.})$$

Розрахуємо абсолютну ефективність вкладених інвестицій $Е_{абс}$ за формулою:

$$\text{Е}_{\text{абс}} = (\text{ПП} - PV), \text{ (грн.)} \qquad (4.9)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн.;

Приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_1^T \frac{\Delta П_i}{(1+\tau)^t}\ , (грн.) \qquad (4.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн.;

Т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні – 0,2;

t – період часу (в роках) від моменту отримання чистого прибутку до точки «0»;

$$ПП = \frac{469\ 157}{(1+0{,}2)^1} + \frac{364\ 966}{(1+0{,}2)^2} + \frac{360\ 744}{(1+0{,}2)^3} = 795\ 323\ (грн)$$

$$Е_{абс} = (795\ 323 - 347\ 688) = 447\ 635\ (грн).$$

Оскільки $Е_{абс} > 0$, результат від проведення наукових досліджень щодо розробки програмного продукту та їх впровадження принесе прибуток, тобто є доцільним, проте це ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні даної програми.

5-й крок. Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1\ , \qquad (4.10)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн.;

PV – теперішня вартість інвестицій PV = ЗВ, грн.

Тж – життєвий цикл наукової розробки, роки.

$$\mathrm{E_в} = \sqrt[3]{1 + \frac{447\,635}{347\,688}} - 1 = 0{,}32 = 32\%.$$

Порівняємо $E_в$ з мінімальною (бар'єрною) ставкою дисконтування τ_{min}, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть.

Спрогнозуємо величину τ_{min}.

У загальному вигляді мінімальна (бар'єрна) ставка дисконтування τ_{min} визначається за формулою:

$$\tau = d + f\,, \qquad (4.11)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; d =(0,14…0,2)

f – показник, що характеризує ризикованість вкладень;

величина f = (0,05…0,1)

.

$$\tau_{min} = 0{,}17 + 0{,}05 = 0{,}22$$

Оскільки $E_в$ = 32% > τ_{min} = 22%, то у інвестора є потенційна зацікавленість у фінансуванні даної наукової розробки.

6-й крок. Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій $\mathrm{T_{ок}}$ за формулою:

$$\mathrm{T_{ок}} = \frac{1}{\mathrm{E_в}}\ \text{рік} \qquad (4.12)$$

$$\mathrm{T_{ок}} = \frac{1}{0{,}32} = 3{,}1\ \text{(роки)}.$$

Оскільки термін окупності вкладених у реалізацію наукового проекту інвестицій 3,1 роки ($Т_{ок}$ < 3…5-ти років), то фінансування нової розробки є доцільним.

4.5 Висновки до розділу

В даному розділі було виконано оцінювання комерційного потенціалу розробки програмного засобу для вдосконалення методу автентифікації у системах контролю версій вихідного коду програмного забезпечення з використанням цифрового підписування.

Проведено технологічний аудит з залученням трьох незалежних експертів. Визначено, що рівень комерційного потенціалу розробки вище середнього. Проведено порівняння з аналогом.

Згідно з проведеним оцінюванням нова розробка є якісною та конкурентоспроможною.

Рівень комерційного потенціалу розробки, становить 41 бал, що відповідає рівню «високий».

Згідно із розрахунками всіх статей витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи загальні витрати на розробку складають 173 844 (грн.). Розрахована абсолютна ефективність вкладених інвестицій в сумі 447 635 (грн.) свідчить про отримання прибутку інвестором від комерціалізації програмного продукту.

Щорічна ефективність вкладених в наукову розробку інвестицій складає 32%, що вище за мінімальну бар'єрну ставку дисконтування, яка складає 22%. Це означає потенційну зацікавленість інвесторів у фінансуванні розробки.

Термін окупності вкладених у реалізацію проекту інвестицій становить 3,1 (роки), що також свідчить про доцільність фінансування нової розробки.

Отже, проаналізувавши отримані економічні показники, можна вважати, що запропонована розробка програмного засобу має високий комерційний потенціал, а тому є доцільною для подальшого впровадження.

ВИСНОВКИ

Стрімкий розвиток інформаційних технологій тягне за собою також зростання складності кіберзагроз, у тому числі тих, що націлені на втручання в роботу систем контролю версій. Це обумовлює потребу переглянути процес автентифікації, визначити вразливості вже наявних методів та впровадити удосконалений метод автентифікації. У ході даної дипломної роботи було проведено розробку програмного забезпечення, що являє собою удосконалений метод автентифікації у системах контролю версій на основі використанні цифрового підписування.

У першому розділі було окреслено сутність поняття автентифікації, розглянуто різні типи кібератак, які можуть застосовуватись під час процесу автентифікації та запропоновано рішення до кожної з них. Також було розглянуто поняття систем контролю версій, розглянуто їх різновидність та необхідність їх використання. Проведено аналіз існуючих методів автентифікації у системах контролю версій. Розглянуто вразливості сучасних систем контролю версій та обумовлено потребу у використанні вдосконалених методів автентифікації.

У другому розділі було здійснено проектування алгоритму автентифікації на основі цифрового підписування. Досліджено можливості застосування цифрового підпису у системах контролю версій для підвищення безпеки. В ході роботи над другим розділом було спроектовано алгоритм автентифікації з використанням цифрового підпису. Окрім цього було здійснено аналіз рівня захисту системи контролю версій з використанням вдосконаленого методу автентифікації та відповідність міжнародним стандартам ISO27001, а також рекомендаціям NIST та CIS Controls. Запропонований удосконалений метод автентифікації відповідає вимогам міжнародних стандартів кібербезпеки, таких як ISO27001, NIST та CIS Controls, що гарантує його відповідність сучасним вимогам щодо захисту даних та кібербезпеки. Ці переваги роблять систему не лише більш безпечною, а й більш ефективною для інтеграції в сучасні процеси розробки програмного забезпечення, що включають автоматизацію перевірок і гарантує безпеку на кожному етапі.

Третій розділ присвячено практичній реалізації програмного забезпечення та аналізу отриманих результатів. Здійснено розробку клієнтської та серверної

частини програмного додатку для вдосконалення методу авторизації у системах контролю версій вихідного коду з використанням цифрового підпису. Окрім цього окреслено чіткі інструкції з використання програмного забезпечення, а також проведено комплексне тестування для перевірки реалізованого програмного забезпечення. На основі практичної реалізації та проведеного аналізу зроблено висновок, що програмне забезпечення гарантує безпеку в процесі автентифікації в системах контролю версій за допомогою цифрового підписування.

У четвертому розділі роботи здійснено аналіз економічної доцільності розробки та впровадження запропонованого удосконаленого методу автентифікації в системах контролю версій на основі цифрового підписування. Отримані в ході дослідження показники свідчать про те, що запропоноване програмне забезпечення має високий комерційний потенціал, а тому є доцільним для подальшого використання.

Опираючись на усі проведенні дослідження та враховуючи наведені результати роботи, можна вважати, що в даній магістерській дипломній роботі вдалось досягнути поставленої мети та виконати усі поставлені задачі, а саме – розроблено та реалізовано програмне забезпечення, що має на меті удосконалити процес автентифікації у системах контролю версій програмного забезпечення за допомогою цифрового підписування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. М. В. Луканов, В. В. Карпінець. Цифровий підпис як метод автентифікації в системах контролю версій вихідного коду програмного забезпечення. Молодь в науці: дослідження, проблеми, перспективи (МН-2025), Вінниця, ВНТУ.

2. Skott Chacon, Ben Straub. Pro Git. 2nd Edition : Apress, 2014

3. Стан, досягнення та перспективи інформаційних систем і технологій. Матеріали XXIV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. Одеса, 18-19 квітня 2024 р. - Одеса, 2024 р. – 498 с.

4. What is access control?. ISO. URL: https://www.iso.org/ru/information-security/%20access-control (дата звернення: 02.10.2024).

5. Коміт у контексті програмування. Tseivo. URL: https://tseivo.com/b/memecode/t/pn8oalbyxz/shcho-take-komit-commit-u-konteksti-prohramuvannia-ta-scm-git (дата звернення: 02.10.2024).

6. About branches. GitHub. URL: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches (дата звернення: 02.10.2024).

7. Ruparel V. RCS−Revision Control System by Walter F. Tichy. Medium. URL: https://medium.com/@viralruparel/rcs-revision-control-system-by-walter-f-tichy-2786b0cb4691 (дата звернення: 02.10.2024).

8. Derek Robert Price, Ximbiot. CVS - Concurrent Versions System] / Derek Robert Price & Ximbiot. – 2019. – URL: : https://cvs.nongnu.org/ (дата звернення: 30.09.2024)

9. Git. *Git*. URL: https://git-scm.com/ (дата звернення: 30.09.2024).

10. GitHub's Success Stories. *GitHub*. URL: https://github.com/customer-stories/all (дата звернення: 02.10.2024).

11. А. Чунарьова, А. Чунарьов. Аналіз існуючих шаблонів систем автентифікації в інформаційно-комунікаційних системах та мережах. *Безпека інформації*. 2012. № 2. С. 65-70.

12. Phishing on GitHub. X*orlab*. URL: https://www.xorlab.com/en/blog/phishing-on-github (дата звернення: 02.10.2024).

13. Мілінчук Ю.А., Жукова О.А. Аналіз загроз процесам автентифікації. Національний технічний університет «Дніпровська політехніка». С. 64–66.

14. Коробейнікова, Т., Непийвода, М., & Матвійчук, А. (2024). Review of solutions for improving authentication and authorization processes on web resources. *SWorldJournal*, *1*(25-01), 82–88. https://doi.org/10.30888/2663-5712.2024-25-00-022x27

15. Що таке автентифікація? Your request has been blocked. This could be due to several reasons. URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-authentication (дата звернення: 30.10.2024).

16. Герт Крюгер. Стандарти інформаційної безпеки. 2022. URL: https://www.dqsglobal.com/uk-ua/navchajtesya/blog/standarti-informacijnoyi-bezpeki-oglyad (дата звернення: 30.10.2024)

17. Holmes S. Getting MEAN with Mongo, Express, Angular, and Node. Manning Publications, 2015. 440 p.

18. К. В. Неуйміна, Ю. В. Баришев. Методи автентифікації віддалених користувачів. *Репозитарій Вінницького Національного Технічного Університету*. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/10947/833.pdf?sequence=3 (дата звернення: 30.10.2024).

19. Сомов, С.В., Канівець, В.Г. (2018). Автентифікація користувачів комп'ютерних систем. Новітні інформаційні системи та технології.

20. Бондаренко О. В., В. В. Карпінець. Двофакторна аутентифікація в системах контролю і управління доступом. *Тези доповідей «Молодь в науці: дослідження, проблеми, перспективи» (МН-2020)*, м. Вінниця, 18-29 травня 2020 р.

21. Онищенко Ю. М., К. К. Петрова. Двофакторна автентифікація, як засіб захисту від несанкціонованого доступу. Зб. матеріалів Всеукр. наук.-практ. конф. (м. Харків, 15 листоп. 2017 р.) МВС України, Харків : ХНУВС, 2017. – С. 146 - 148.

22. Костікова М. В., Гетало Д. І., Гура В. О. Комп'ютерна безпека – цілісність даних і конфіденційність інформації. *Матеріали 84-ї міжнародної студентської наукової конференції ХНАДУ*. 2022.

23. SSH Key Pair Explained: How SSH Private & Public Keys Work. Sectigo® Official. URL: https://www.sectigo.com/resource-library/what-is-an-ssh-key (дата звернення: 30.09.2024).

24. Д. М. Жаворонок. Метод автентифікації користувача за допомогою токенів. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/33651/Жаворонок.pdf?sequence=1&isAllowed=y (дата звернення: 30.10.2024)

25. Здолбіцька, Н., Жигаревич, О. Сучасні способи автентифікації та захист на основі токенів. *Прикладні проблеми комп'ютерних наук, безпеки та математики*, № (3), с. 4-11. 2024.

26. Коваленко Ю.Б., Орлик О. В. Digital signature and distinctive features of its usage. 2015. С. 16–19.

27. Закон України від 22.05.2003 р. №851-IV «Про електронні документи та електронний документообіг»

28. Закон України від 22.05.2003 р. №852-IV «Про електронний цифровий підпис»

29. Ореахов І., Комих Н. Пріорітети забезпечення кібербезпеки України. Матеріали VIII Міжнародної науково-практичної конференції (ДДУВС, 15.03.2024). Частина ІІ. 2024.

30. Єсін В.І., Вілігура В. В, Узлов Д. Ю. Огляд існуючих моделей та основних принципів нульової довіри. Радіотехніка, № 217. 2024.

31. Laura Dabbish, Colleen Stuart. Social coding in GitHub. CSCW '12: Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work

32. About MySQL. MySQL. URL: https://www.mysql.com/about/ (дата звернення: 30.10.2024).

33. About Python. Python. URL: https://www.python.org/about/ (дата звернення: 30.10.2024).

34. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ

Голова секції "Управління інформаційною безпекою" кафедри МБІС

д.т.н., професор

_____ Юрій ЯРЕМЧУК

"20" вересня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:

Вдосконалення методу автентифікації у системах контролю версій вихідного коду програмного забезпечення з використанням цифрового підписування

08-72.МКР.003.00.000.ТЗ

Керівник магістерської кваліфікаційної роботи

к.т.н., доцент

_____ Карпінець В. В.

"____" _____

Вінниця – 2024 р.

1. Найменування та область застосування

Система вдосконаленої автентифікації у системах контролю версій вихідного коду програмного забезпечення з використанням цифрового підпису.

2. Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ №310 від 17.09.2024 р.

3. Мета та призначення розробки

3.1 Мета розробки: створення алгоритму та програмного забезпечення для вдосконалення методів автентифікації у системах контролю версій із застосуванням цифрового підпису.

3.2 Призначення: забезпечення надійного контролю цілісності вихідного коду в системах контролю версій, виявлення несанкціонованих змін та захисту від підробки.

4. Джерела розробки

4.1. Rivest, R., Shamir, A., & Adleman, L. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM.

4.2. NIST SP 800-63 (2021). Digital Identity Guidelines.

4.3. ISO/IEC 27001:2022. Information Security Management Systems – Requirements.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програма має реалізовувати генерацію цифрових підписів, перевірку автентичності та контроль цілісності вихідного коду.

5.1.2 Інтерфейс користувача має бути зручним, інтуїтивно зрозумілим.

5.1.3 Має бути забезпечена інтеграція з системами контролю версій вихідного коду програмного забезпечення.

5.2 Вимоги до надійності:

5.2.1 Програмне забезпечення повинно стабільно функціонувати в середовищах командної розробки

5.2.2 Підтримка перевірки підписів на різних етапах життєвого циклу програмного забезпечення.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор –Intel Core i3 або еквівалент;

– оперативна пам’ять – не менше 1 GB;

– середовище функціонування – операційна система Windows, Linux або MacOS;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні технікою.

6. Вимоги до програмної документації

6.1 Інструкція користувача повинна містити опис процесів генерації ключів, підпису файлів, перевірки автентичності та інтеграції з системами CI/CD.

7. Вимоги до технічного захисту інформації

7.1 Застосування криптографічних механізмів для захисту ключів від компрометації.

8. Техніко-економічні показники

8.1 Впровадження розробки підвищує рівень безпеки вихідного коду та дозволяє знизити ризики компрометації.

8.2 Очікується скорочення витрат на усунення наслідків несанкціонованих змін коду та підтримку відповідності стандартам з кібербезпеки.

9. Стадії та етапи розробки

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напрямку магістерської кваліфікаційної роботи, формулювання теми	20.09.2024	25.09.2024	
2	Аналіз предметної області обраної теми	26.09.2024	05.10.2024	
3	Розробка алгоритму роботи	06.10.2024	16.10.2024	
4	Написання магістерської кваліфікаційної роботи на основі розробленої теми	17.10.2024	20.11.2024	
5	Передзахист магістерської кваліфікаційної роботи	21.11.2024	25.11.2024	
6	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	26.11.2024	08.12.2024	
7	Захист магістерської кваліфікаційної роботи	10.12.2024	11.12.2024	

10. Порядок контролю та прийому

10.1 До приймання магістерської дипломної роботи надається:

- ПЗ до магістерської дипломної роботи;
- демонстрація результату магістерської дипломної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____Луканов М. В.

Додаток Б. Лістинг коду генерації ключів

```
import os

import hashlib

from ecdsa import SigningKey, NIST384p

def generate_keys():

    private_key = SigningKey.generate(curve=NIST384p)

  public_key = private_key.verifying_key

  return private_key, public_key

def save_key_to_file(key, file_path):

  with open(file_path, 'wb') as file:

    file.write(key.to_pem())

  print(f"Ключ збережено у: {file_path}")

def main():

  print("Генерація ключів для цифрового підпису")

  first_name = input("Введіть ім'я: ")

  last_name = input("Введіть прізвище: ")

    private_key, public_key = generate_keys()

  save_path = input("Введіть шлях для збереження ключів (натисніть Enter для поточної директорії): ")

  if not save_path:

    save_path = os.getcwd()

    private_key_filename = f"{first_name}_{last_name}_private_key.pem"

  public_key_filename = f"{first_name}_{last_name}_public_key.pem"

    private_key_path = os.path.join(save_path, private_key_filename)

  public_key_path = os.path.join(save_path, public_key_filename)
```

```
save_key_to_file(private_key, private_key_path)
    save_key_to_file(public_key, public_key_path)
if __name__ == "__main__":
main()
```

Додаток В. Лістинг коду накладання підпису

```
import os
import hashlib
import requests
from ecdsa import SigningKey, NIST384p
def generate_file_hash(file_path):
    sha256_hash = hashlib.sha256()
    with open(file_path, 'rb') as file:
        for byte_block in iter(lambda: file.read(4096), b""):
            sha256_hash.update(byte_block)
    return sha256_hash.hexdigest()
def sign_data(private_key_path, data):
    with open(private_key_path, 'rb') as key_file:
        private_key = SigningKey.from_pem(key_file.read())
    signature = private_key.sign(data.encode(), hashfunc=hashlib.sha256)
    return signature.hex()
def main():
    private_key_path = input("Введіть шлях до приватного ключа: ")
    if not os.path.exists(private_key_path):
        print("Невірний шлях до приватного ключа.")
        return
    directory = input("Введіть шлях до директорії з файлами (натисніть Enter для поточної): ")
    if not directory:
        directory = os.getcwd()
    for dirpath, dirnames, filenames in os.walk(directory):
        dirnames[:] = [d for d in dirnames if d != '.git']

        for filename in filenames:
            if '.git' in dirpath:
                continue
            file_path = os.path.join(dirpath, filename)
            if os.path.isfile(file_path):
```

```
            file_hash = generate_file_hash(file_path)
            signature = sign_data(private_key_path, file_hash)
            print(f"Підписуємо та відправляємо файл: {filename}")
        response = requests.post('http://127.0.0.1:5000/upload', json={
            'filename': filename,
            'file_hash': file_hash,
            'signature': signature
        })
        if response.status_code == 201 or response.status_code == 200:
            print(f"Файл '{filename}' успішно відправлено на сервер.")
        else:
            print(f"Помилка відправки файлу '{filename}': {response.json().get('error', 'Unknown
error')}")
if __name__ == "__main__":
    main()
```

Додаток Г. Лістинг коду верифікації підпису

```
import os
import hashlib
import requests
import sys
def generate_file_hash(file_path):
    sha256_hash = hashlib.sha256()
    with open(file_path, 'rb') as file:
        for byte_block in iter(lambda: file.read(4096), b""):
            sha256_hash.update(byte_block)
    return sha256_hash.hexdigest()
def verify_file_on_server(filename, file_path):
    file_hash = generate_file_hash(file_path)
    response = requests.post('http://127.0.0.1:5000/verify', json={
        'filename': filename,
        'file_hash': file_hash,
        'file_path': file_path
    })
    if response.status_code == 200:
        print(f"Верифікація файлу '{filename}' пройшла успішно.")
        return True
    else:
        print(f"Помилка верифікації файлу '{filename}': {response.json()['error']}")
        return False
def main():
    error_occurred = False
    directory = input("Введіть шлях до директорії для верифікації (натисніть Enter для
поточної): ")
    if not directory:
        directory = os.getcwd()
    for dirpath, dirnames, filenames in os.walk(directory):
        dirnames[:] = [d for d in dirnames if d != '.git']
```

```
        for filename in filenames:
            if '.git' in dirpath:
                continue
            file_path = os.path.join(dirpath, filename)
            if os.path.isfile(file_path):
                print(f"Перевірка файлу: {filename}")
                if not verify_file_on_server(filename, file_path):
                    error_occurred = True
    if error_occurred:
        print("Є несанкціоновані зміни!")
        sys.exit(1)
    print("Всі файли були верифіковані успішно.")
if __name__ == "__main__":
    main()
```

Додаток Д. Лістинг коду сервера

```
from flask import Flask, request, jsonify
import mysql.connector
from ecdsa import VerifyingKey, BadSignatureError
import hashlib
import os
app = Flask(__name__)
db_config = {
    'host': os.getenv('DB_HOST'),
    'user': os.getenv('DB_USER'),
    'password': os.getenv('DB_PASSWORD'),
    'database': os.getenv('DB_NAME')
}
def connect_db():
    return mysql.connector.connect(**db_config)
def verify_signature(public_key, signature, file_hash):
    try:
        vk = VerifyingKey.from_pem(public_key)
        vk.verify(bytes.fromhex(signature), file_hash.encode(), hashfunc=hashlib.sha256)
        return True
    except BadSignatureError:
        return False
@app.route('/upload', methods=['POST'])
def upload_file():
    try:
        data = request.get_json()
        filename = data['filename']
        file_hash = data['file_hash']
        signature = data['signature']
        conn = connect_db()
        cursor = conn.cursor()
        cursor.execute("SELECT public_key FROM trusted_public_keys LIMIT 1")
```

```
        result = cursor.fetchone()
        if result is None:
            return jsonify({'error': 'Public key not found'}), 404
        public_key = result[0]
        if not verify_signature(public_key, signature, file_hash):
            return jsonify({'error': 'Invalid signature'}), 400
        cursor.execute("SELECT * FROM files WHERE filename = %s", (filename,))
        existing_file = cursor.fetchone()
        if existing_file:
            update_query = """
            UPDATE files
            SET digital_signature = %s, public_key = %s, file_hash = %s
            WHERE filename = %s
            """
            cursor.execute(update_query, (signature, public_key, file_hash, filename))
            conn.commit()
            return jsonify({'message': 'File metadata updated successfully'}), 200
        else:
            insert_query = """
            INSERT INTO files (filename, file_hash, digital_signature, public_key)
            VALUES (%s, %s, %s, %s)
            """
            cursor.execute(insert_query, (filename, file_hash, signature, public_key))
            conn.commit()
            return jsonify({'message': 'File metadata saved successfully'}), 201
    except Exception as e:
        return jsonify({'error': str(e)}), 500
@app.route('/verify', methods=['POST'])
def verify_file():
    try:
        data = request.json
        filename = data['filename']
```

```
    file_path = data.get('file_path')
    current_hash = data.get('file_hash')
    conn = connect_db()
    cursor = conn.cursor(dictionary=True)
    select_query = "SELECT * FROM files WHERE filename = %s"
    cursor.execute(select_query, (filename,))
    file_record = cursor.fetchone()
    if not file_record:
        ip_address = request.remote_addr
        cursor.execute(
            "INSERT INTO incidents (filename, issue, ip_address) VALUES (%s, %s, %s)",
            (filename, 'File metadata not found', ip_address)
        )
        conn.commit()
        return jsonify({'error': 'File metadata not found in the database'}), 404
    if current_hash != file_record['file_hash']:
        ip_address = request.remote_addr
        cursor.execute(
            "INSERT INTO incidents (filename, issue, ip_address) VALUES (%s, %s, %s)",
            (filename, 'File hash mismatch', ip_address)
        )
        conn.commit()
        return jsonify({'error': 'File hash mismatch'}), 400
    if not verify_signature(file_record['public_key'], file_record['digital_signature'],
file_record['file_hash']):
        ip_address = request.remote_addr
        cursor.execute(
            "INSERT INTO incidents (filename, issue, ip_address) VALUES (%s, %s, %s)",
            (filename, 'Signature verification failed', ip_address)
        )
        conn.commit()
        return jsonify({'error': 'Signature verification failed'}), 400
```

```
        return jsonify({'message': 'File verification successful'}), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Додаток Е. Ілюстративний матеріал

Вдосконалення методу автентифікації у системах контролю версій вихідного коду програмного забезпечення з використанням цифрового підписування

Виконав: ст. групи КІТС-23м Луканов М.В.
Керівник: к.т.н., доц., доцент каф. МБІС Карпінець В.В.

Актуальність

Використання цифрового підпису для автентифікації у системах контролю версій є перспективним рішенням, що забезпечує цілісність програмного коду та захист від кіберзагроз, мінімізуючи ризики втручання та несанкціонованих зміни.

Мета

Метою даної магістерської роботи є розробка методу автентифікації для систем контролю версій, що забезпечить підвищення безпеки та цілісності програмного коду за рахунок використання цифрового підпису, а також його практична реалізація.

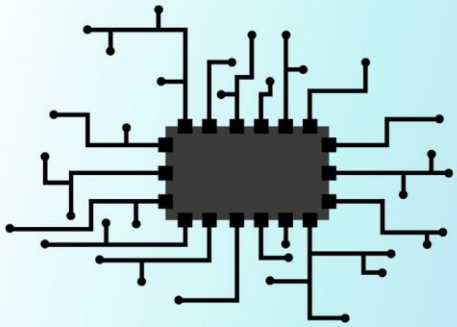
Завдання

– проаналізувати виклики, ризики та існуючі методи автентифікації у системах контролю версій;
– вивчити можливості застосування цифрового підпису для підвищення безпеки;
– спроєктувати удосконалений метод автентифікації з використанням цифрового підпису;
– реалізувати запропонований метод у вигляді програмного забезпечення та провести його тестування;

CI/CD — це автоматизований підхід до інтеграції та доставки коду, який використовує робочі процеси для побудови, тестування та розгортання програмного забезпечення, забезпечуючи швидкість, надійність і мінімізацію людського втручання.

Що таке система контролю версій?

Система контролю версій — це технологія, яка забезпечує збереження, управління та відстеження змін у коді дозволяючи організовувати співпрацю між кількома користувачами, уникати конфліктів і зберігати історію змін для подальшого аналізу чи відновлення.



Цифровий підпис — це криптографічний механізм, який забезпечує підтвердження автентичності та цілісності даних у цифровій формі, підтверджуючи відсутність неавтентифікованих змін і захищаючи інформацію від підробки.

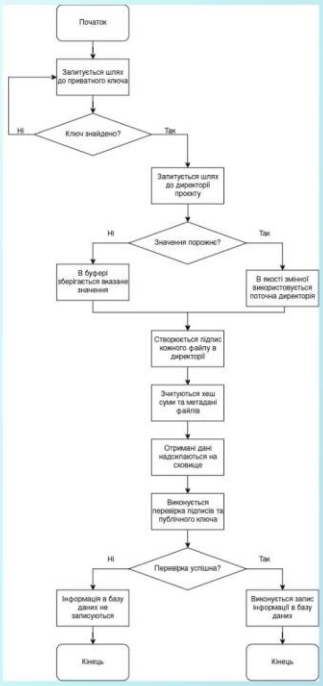
Надійна автентифікація забезпечує захист від несанкціонованого змін в системах контролю версій вихідного коду та процесах CI/CD.

Моніторинг таких інцидентів дозволяє швидко виявляти загрози та оперативно реагувати на порушення, забезпечуючи високий рівень безпеки в процесах розробки, розгортання та підтримки IT проектів.

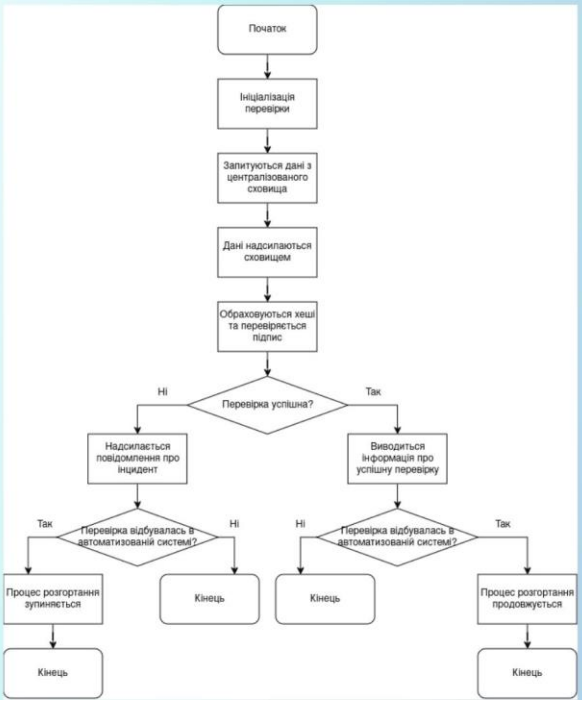
Порівняння рівня захисту

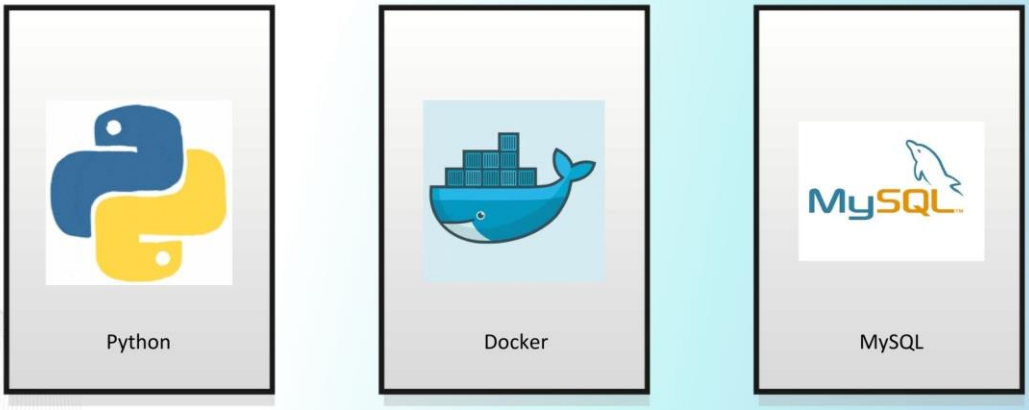
	Пароль	Двофакторна автентифікація	SSH-ключі	Токени доступу	Вдосконалений метод з цифровим підписом
Несанкціонований доступ через викрадені облікові дані	Захист слабкий, можливе використання скомпрометованих паролів	Стійкий до компрометації паролів завдяки додатковому фактору.	Високий рівень захисту, якщо приватний ключ зберігається безпечно.	Високий рівень, якщо токен зберігається у захищеному середовищі.	Забезпечується через обов'язковий процес перевірки підпису.
Підробка змін у коді	Не захищає.	Не захищає.	Не захищає.	Не захищає.	Підпис кожного файлу забезпечує виявлення змін.
Компрометація ключів/токенів	Немає ключів або токенів.	Немає ключів або токенів.	Уразливий, якщо приватний ключ зкомпрометовано.	Уразливий, якщо токен вкрадено.	Забезпечує додатковий фактор захисту. Ключ підпису не надає доступ до репозиторію
Недосконалості розробленого коду	Не захищає.	Не захищає.	Не захищає.	Не захищає.	Регулююча особа схвалює зміни перед підписом.
Соціальна інженерія	Захист слабкий.	Захист середній, через фішинговий сайт можливо отримати код з другого фактору	Уразливий, якщо приватний ключ не захищений.	Уразливий, якщо токен отримано зловмисником.	Приватний ключ не поширюється підписантом та використовується лише у визначених програмах

Процес накладання підпису



Блок-схема алгоритму перевірки підпису





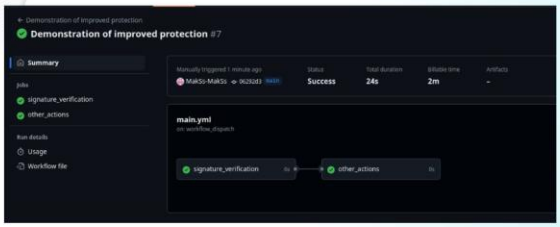
Генерація ключа регулюючої особи

```
→  MKR git:(main) python3 keygen.py
Генерація ключів для цифрового підпису
Введіть ім'я: Maksym
Введіть прізвище: Lukanov
Введіть шлях для збереження ключів (натисніть Enter для поточної директорії):
Ключ збережено у: /home/maksym/MKR/Maksym_Lukanov_private_key.pem
Ключ збережено у: /home/maksym/MKR/Maksym_Lukanov_public_key.pem
→  MKR git:(main) ✗
```

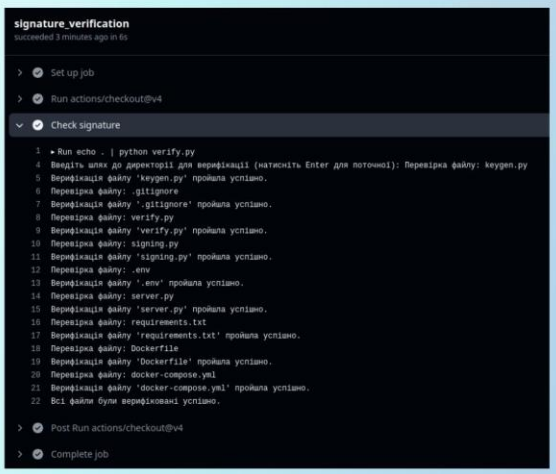
Підпис файлів

```
Введіть шлях до приватного ключа: ../keys_for_MKR/Maksym_Lukanov_private_key.pem
Введіть шлях до директорії з файлами (натисніть Enter для поточної):
Підписуємо та відправляємо файл: keygen.py
Файл 'keygen.py' успішно відправлено на сервер.
Підписуємо та відправляємо файл: .gitignore
Файл '.gitignore' успішно відправлено на сервер.
Підписуємо та відправляємо файл: verify.py
Файл 'verify.py' успішно відправлено на сервер.
Підписуємо та відправляємо файл: signing.py
Файл 'signing.py' успішно відправлено на сервер.
Підписуємо та відправляємо файл: .env
Файл '.env' успішно відправлено на сервер.
Підписуємо та відправляємо файл: server.py
Файл 'server.py' успішно відправлено на сервер.
Підписуємо та відправляємо файл: docker-compose.yml
Файл 'docker-compose.yml' успішно відправлено на сервер.
Підписуємо та відправляємо файл: requirements.txt
Файл 'requirements.txt' успішно відправлено на сервер.
Підписуємо та відправляємо файл: Dockerfile
Файл 'Dockerfile' успішно відправлено на сервер.
→  MKR git:(main)
```

Інтеграція в CI/CD



Виконання перевірки під час CI/CD



Приклад ідентифікації неавтентифікованих змін

```
✗ MKR git:(main) echo "#test" >> keygen.py
✗ MKR git:(main) ✗ touch test.file
✗ MKR git:(main) ✗ python3 verify.py
Введіть шлях до директорії для верифікації (натисніть Enter для поточної):
Перевірка файлу: keygen.py
Помилка верифікації файлу 'keygen.py': File hash mismatch
Перевірка файлу: .gitignore
Верифікація файлу '.gitignore' пройшла успішно.
Перевірка файлу: test.file
Помилка верифікації файлу 'test.file': File metadata not found in the database
Перевірка файлу: verify.py
Верифікація файлу 'verify.py' пройшла успішно.
Перевірка файлу: signing.py
Верифікація файлу 'signing.py' пройшла успішно.
Перевірка файлу: .env
Верифікація файлу '.env' пройшла успішно.
Перевірка файлу: server.py
Верифікація файлу 'server.py' пройшла успішно.
Перевірка файлу: docker-compose.yml
Верифікація файлу 'docker-compose.yml' пройшла успішно.
Перевірка файлу: requirements.txt
Верифікація файлу 'requirements.txt' пройшла успішно.
Перевірка файлу: Dockerfile
Верифікація файлу 'Dockerfile' пройшла успішно.
Є несанкціоновані зміни!
✗ MKR git:(main) ✗
```

Фіксування інциденту для SIEM та SOAR

id	filename	issue	ip_address	created_at ▾ 1
71	keygen.py	Signature verification failed	[redacted]	2024-12-[redacted]
68	keygen.py	File hash mismatch	[redacted]	2024-12-[redacted]
69	test.file	File metadata not found	[redacted]	2024-12-[redacted]

Споживання ресурсів

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O
015f67568159	server-flask-1	0.02%	27.84MiB / 1.918GiB	1.42%	1.31MB / 813kB	3.88MB / 1.72MB
a959c5bbfae3	server-mysql-1	0.07%	185.5MiB / 1.918GiB	9.45%	2.22MB / 2.91MB	1.17MB / 345MB

Низькі системні вимоги надають можливість впроваджувати рішення для підприємств з обмеженим бюджетом або складними умовами інтеграції нових систем.

РЕЗУЛЬТАТИ РОБОТИ

В даній магістерській дипломній роботі вдалося досягнути поставленої мети та виконати усі задачі, а саме – вдосконаленно метод автентифікації в системах контролю версій, шляхом інтеграції цифрового підписування для підтвердження автентичності та цілісності змін у вихідному коді програмного забезпечення. Розроблене рішення забезпечує надійний захист від несанкціонованих змін і підвищує рівень безпеки в процесах розробки.

Використання цифрових підписів дозволяє ефективно мінімізувати ризики неавтентифікованих змін. Результати аналізу підтвердили, що впроваджена система автентифікації захищає від більшого спектра атак, в порівнянні з існуючими методами, забезпечуючи високу цілісність і безпеку коду в процесах контролю версій та підтримки існуючих проектів.

ДЯКУЮ ЗА УВАГУ!

Додаток Ж. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:
Вдосконалення методу автентифікації у системах контролю версій вихідного коду програмного забезпечення з використанням цифрового підписування

Тип роботи: магістерська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. КІТС-23м

Керівник доцент Карпінець В.В.

Показники звіту подібності

Turnitin	
Оригінальність	88%
Загальна схожість	12%

Аналіз звіту подібності (відмітити потрібне)

- ☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Луканов М.В.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)