Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Архітектура та протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi
Назва теми

Рівень вищої освіти *другий (магістерський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр КвРКІ 024018.24.01.03 ПЗ

Виконав здобувач ІІ курсу, група КІ2м-24-1 _____ Вадим ГУРСЬКИЙ
Підпис Ініціали, прізвище

Керівник доцент канд.-техн. наук, _____ Катерина БЕРЕЗЬКА
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер д. техн. наук, професор _____ Сергій ЛИСЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС _____ Ольга ПАВЛОВА
«___» _травня_ 2026 р. підпис Ініціали, прізвище
дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень МАГІСТР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма ОСВІТНЬО-НАУКОВА ПРОГРАМА «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Ольга ПАВЛОВА

"10" 01 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Вадиму ГУРСЬКОМУ
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Архітектура та протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi

Керівник проекту (роботи) Катерина БЕРЕЗЬКА, к.т.н., доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 12.01.2026 р. № 6

2. Строк подання студентом проекту (роботи) на кафедру 01.05.2026 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Процес верифікації та аналіз відомих методів та засобів віддаленої верифікації Kubernetes pod'ів для edge-пристроїв

Модель процесу віддаленої верифікації Kubernetes подів

Архітектура віддаленої верифікації Kubernetes pod'ів для edge-пристроїв на базі Raspberry Pi

Розгортання та експериментальні дослідження протоколу віддаленої верифікації Kubernetes pod'ів для edge-пристроїв на базі Raspberry Pi

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів кваліфікаційної роботи магістра

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «_12_»_____01_____2026р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи магістра	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики КвРМ з керівником	10.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	12.01.2026	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	15.01.2026	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	15.02.2026	виконано
5	Робота над науковою статтею	5.03.2026	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.03.2026	виконано
7	Робота над розділом 4 – проектування та розгортання пропонованого рішення, експериментальна частина	15.04.2026	виконано
8	Оформлення пояснювальної записки згідно вимог	25.04.2026	виконано
9	Попередній захист ДРМ	29.04.2026	виконано
10	Захист ДРМ на засіданні ЕК	До 15.05.2026	

Здобувач _____ Вадим ГУРСЬКИЙ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Катерина БЕРЕЗЬКА
Підпис Ім'я, ПРІЗВИЩЕ

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: Архітектура та протокол віддаленої верифікації Kubernetes Pod’ів для edge-пристроїв на базі Raspberry Pi

Автор роботи: Гурський Вадим Юрійович

Керівник роботи: Березька Катерина Миколаївна

Пояснювальна записка: 84 с., 11 рис., 0 табл., 5 дод., 76 джерел.

ВІДДАЛЕНА ВЕРИФІКАЦІЯ, КЛАСТЕР, EDGE ПРИСТРОЇ, ОДНОПЛАТНА КОМП’ЮТЕРНА СИСТЕМА RASPBERRY PI

Об’єктом дослідження є процеси віддаленої верифікації робочих вузлів та окремих Kubernetes Pod’ів у розподілених edge-кластерах на базі Raspberry Pi з використанням апаратного TPM-модуля.

Предметом дослідження є методи, засоби та фреймворки віддаленої верифікації робочих вузлів та окремих Kubernetes Pod’ів у розподілених edge-кластерах.

Метою кваліфікаційної роботи магістра є проведення перевірки цілісності на рівні окремих Pod’ів у розподілених кластерах з обмеженими обчислювальними ресурсами, а також зменшення часу проведення процесу верифікації шляхом дослідження та проєктування архітектури й протоколу віддаленої атестації Kubernetes Pod’ів для edge-пристроїв на базі Raspberry Pi з використанням апаратного TPM-модуля Infineon OPTIGA SLB 9670.

Для розв’язання поставлених задач використовувалися методи теорії множин, формального моделювання станів і переходів систем, криптографічного аналізу протоколів, системного аналізу архітектур розподілених обчислень.

Наукова новизна отриманих результатів:

– набув подальшого розвитку протокол віддаленої верифікації Kubernetes Pod’ів для edge-пристроїв на базі Raspberry Pi, який дозволяє забезпечити гранулярну перевірку цілісності окремих контейнерів у розподілених кластерах з обмеженими ресурсами, і який відрізняється від відомих рішень модифікацією підсистеми IMA ядра Linux для додавання атрибута cgroup-path до вимірювання та

інтеграцією із апаратним TPM-модулем типу Infineon OPTIGA SLB 9670 через SPI-інтерфейс, що дозволило зменшити час розгортання системи у порівнянні із відомим інструментом keylime.

– набула подальшого розвитку архітектура віддаленої верифікації Kubernetes pod'ів, яка відрізняється від відомих прямою інтеграцією з апаратним TPM-модулем Infineon OPTIGA SLB 9670 через SPI-інтерфейс на Raspberry Pi, що дозволило використовувати реальний апаратний корінь довіри замість віртуальної емуляції.

На основі проведених досліджень розроблено архітектуру та протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi.

Практична значимість отриманих результатів полягає у розроблених архітектурі та протоколі віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi.

У першому розділі досліджено відомі методи та засоби віддаленої верифікації Kubernetes подів для edge-пристроїв, а також проаналізовано їхні функціональні можливості та обмеження. Проведений аналіз показав, що наявні рішення, зокрема фреймворк Keylime, не забезпечують достатнього рівня гранулярності атестації та не мають тісної нативної інтеграції з Kubernetes API, що ускладнює застосування цих підходів у динамічному контейнеризованому середовищі. У результаті було обґрунтовано доцільність розробки спеціалізованої архітектури віддаленої верифікації, орієнтованої на особливості Kubernetes та експлуатацію edge-пристроїв.

У другому розділі досліджено криптографічні механізми захисту, що ґрунтуються на використанні апаратного модуля довіри для фіксації станів системи та побудови ланцюга довіри. Сформовано теоретичну модель процесу верифікації, яка формалізує взаємодію між компонентами та описує математичні засади перевірки цілісності в розподілених середовищах.

У третьому розділі розроблено архітектуру системи, що інтегрується в платформу через стандартні механізми розширюваності, та формалізовано протоколи реєстрації вузлів і верифікації подів. Запропонований підхід забезпечує

наскрізний контроль від апаратного рівня одноплатного комп’ютера до окремої контейнерної одиниці, дозволяючи оперативно усувати лише скомпрометовані елементи.

У четвертому розділі проведено експериментальну перевірку запропонованого підходу на мінімальному edge-кластері Kubernetes, розгорнутому на одноплатних комп’ютерах Raspberry Pi з апаратним модулем довіри TPM Infineon OPTIGA SLB 9670. Підтверджено коректну інтеграцію TPM через SPI-інтерфейс та готовність платформи до виконання операцій вимірювання і верифікації цілісності.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

AIK - Attestation Identity Key

CA - Certificate Authority

CRD - Custom Resource Definition

CSC - Контролер стану кластера

EK - Endorsement Key

HMAC - Hash-based Message Authentication Code

IaaS - Infrastructure as a Service

IMA - Integrity Measurement Architecture

ML - Measurement Log

PCR - Platform Configuration Register

ВСТУП

Стрімкий розвиток концепції периферійних обчислень та масштабне впровадження edge computing архітектур у промислових, транспортних та IoT системах зумовлює зростаючу потребу у забезпеченні довіри до розподіленої інфраструктури, де обчислювальні ресурси розміщуються географічно близько до джерел даних, часто у фізично незахищених локаціях з обмеженим контролем з боку адміністраторів. Використання Kubernetes як де-факто стандарту оркестрації контейнеризованих застосунків поширилось за межі традиційних центрів обробки даних на edge-пристрої з обмеженими ресурсами, зокрема одноплатні комп'ютери Raspberry Pi, що створює нові виклики у сфері кібербезпеки. Відсутність нативних механізмів віддаленої атестації у Kubernetes залишає критичну прогалину у забезпеченні цілісності виконуваних застосунків, оскільки компрометація периферійного вузла може призвести до витоку конфіденційних даних, маніпуляції результатами обробки інформації або використання інфраструктури для здійснення подальших атак. Особливо гостро ця проблема проявляється у багатоорендному середовищі, де різні організації спільно використовують edge-інфраструктуру хмарного провайдера, вимагаючи надійних криптографічних гарантій того, що їхні застосунки виконуються на довірених вузлах без несанкціонованих модифікацій програмного коду чи конфігурацій.

Існуючі фреймворки віддаленої атестації, зокрема Keylime, хоча і забезпечують верифікацію цілісності фізичних вузлів через апаратні модулі довіри TPM, оперують на рівні цілої операційної системи без можливості розрізнення окремих контейнерів чи Pod'ів, що унеможливлює granular політики безпеки та точкове реагування на компрометацію індивідуальних застосунків. Це обмеження набуває критичного значення у контексті мікросервісних архітектур, де на одному вузлі одночасно можуть виконуватись десятки Pod'ів різних тенантів з різними рівнями довіри та вимогами до безпеки.

Метою кваліфікаційної роботи магістра є проведення перевірки цілісності на рівні окремих Pod’ів у розподілених кластерах з обмеженими обчислювальними

ресурсами, а також зменшення часу проведення процесу верифікації шляхом дослідження та проєктування архітектури й протоколу віддаленої атестації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi з використанням апаратного TPM-модуля Infineon OPTIGA SLB 9670.

Поставлена мета досягається розв'язанням таких основних завдань:

– дослідити процес віддаленої верифікації та пов'язані з ним механізми безпеки;

– проаналізувати відомі методи, засоби та фреймворки віддаленої верифікації комп'ютерних та вузлів кластерних систем, дослідити їх механізми функціонування, виявити їх недоліки та проаналізувати шляхи вдосконалення;

– розробити модель процесу віддаленої верифікації Kubernetes подів для edge-пристроїв;

– розробити архітектуру віддаленої верифікації Kubernetes pod'ів;

– розробити протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi;

– здійснити розгортання кластеру та провести експериментальні дослідження протоколу віддаленої верифікації kubernetes pod'ів для edge-пристроїв на базі raspberry pi;

– провести оцінку ефективності пропонованої архітектури та протоколу віддаленої верифікації через порівняння з базовою конфігурацією Kubernetes кластера без механізмів верифікації, а також з реалізацією на базі фреймворку Keylime.

Об'єктом дослідження є процеси віддаленої верифікації робочих вузлів та окремих Kubernetes Pod'ів у розподілених edge-кластерах на базі Raspberry Pi з використанням апаратного TPM-модуля.

Предметом дослідження є методи, засоби та фреймворки віддаленої верифікації робочих вузлів та окремих Kubernetes Pod'ів у розподілених edge-кластерах.

Наукова новизна отриманих результатів:

– набув подальшого розвитку протокол віддаленої верифікації Kubernetes Pod’ів для edge-пристроїв на базі Raspberry Pi, який дозволяє забезпечити гранулярну перевірку цілісності окремих контейнерів у розподілених кластерах з обмеженими ресурсами, і який відрізняється від відомих рішень модифікацією підсистеми IMA ядра Linux для додавання атрибута cgroup-path до вимірювання та інтеграцією із апаратним TPM-модулем типу Infineon OPTIGA SLB 9670 через SPI-інтерфейс, що дозволило зменшити час розгортання системи у порівнянні із відомим інструментом keylime.

– набула подальшого розвитку архітектура віддаленої верифікації Kubernetes pod’ів, яка відрізняється від відомих прямою інтеграцією з апаратним TPM-модулем Infineon OPTIGA SLB 9670 через SPI-інтерфейс на Raspberry Pi, що дозволило використовувати реальний апаратний корінь довіри замість віртуальної емуляції.

Практична значимість отриманих результатів полягає у розроблених архітектурі та протоколі віддаленої верифікації Kubernetes Pod’ів для edge-пристроїв на базі Raspberry Pi

Для розв’язання поставлених задач використовувалися методи та практики розгортання комп’ютерних кластерів Kubernetes та методи математичного моделювання.

1 ПРОЦЕС ВЕРИФІКАЦІЇ ТА АНАЛІЗ ВІДОМИХ МЕТОДІВ ТА ЗАСОБІВ ВІДДАЛЕНОЇ ВЕРИФІКАЦІЇ KUBERNETES POD'ІВ ДЛЯ EDGE-ПРИСТРОЇВ

1.1 Концепція edge-обчислень та її роль у розподілених системах

Edge-обчислення є парадигмою, що радикально змінює підходи до обробки даних у сучасних інформаційних системах, переносячи обчислювальні ресурси з централізованих хмарних дата-центрів ближче до джерел даних – на периферійні пристрої, такі як сенсори, камери, промислові контролери чи мобільні гаджети [1-5]. На відміну від традиційної хмарної моделі, де всі дані агрегуються та аналізуються в віддалених серверах, edge-обчислення передбачає локальну обробку інформації в реальному часі, що дозволяє зменшити затримки, оптимізувати пропускну здатність мережі та підвищити автономність систем (рис. 1.1). Ця концепція особливо актуальна в умовах зростання обсягів даних від IoT-пристроїв, де щосекунди генеруються терабайти інформації від мільйонів підключених об'єктів, таких як розумні міста, промислові конвеєри чи автономні транспортні засоби. У таких сценаріях централізована обробка стає неефективною через обмежену пропускну здатність каналів зв'язку, ризики перебоїв у мережі та вимоги до миттєвої реакції, наприклад, у системах реального часу для моніторингу обладнання чи ви явлення аномалій у трафіку.

Принцип роботи edge-обчислень базується на ієрархічній архітектурі, де периферійні вузли виконують первинну обробку даних – фільтрацію, агрегацію та базовий аналіз – перед передачею агрегованих результатів до хмарних або гібридних центрів [6-10]. Ключовим елементом цієї ієрархії є шлюзи, які слугують посередниками між edge-пристроями та верхніми рівнями мережі. Шлюзи, часто реалізовані на компактних платформах типу Raspberry Pi чи Intel NUC, агрегують дані з множинних сенсорів, забезпечують протокол-конверсію (наприклад, з MQTT до HTTP) та виконують функції оркестрації, такі як розподіл завдань чи балансування навантаження. У контексті Kubernetes для edge-обчислень шлюзи набувають ролі "edge-оркестраторів", де кластер k3s (полегшена версія Kubernetes)

розгортається на периферії для керування контейнеризованими подами, що виконують мікросервіси. Наприклад, шлюз може оркеструвати поди для локального машинного навчання на даних з камер спостереження, передаючи лише критичні алерти до хмари, що зменшує трафік на 70–80% порівняно з повною передачею сирих даних.

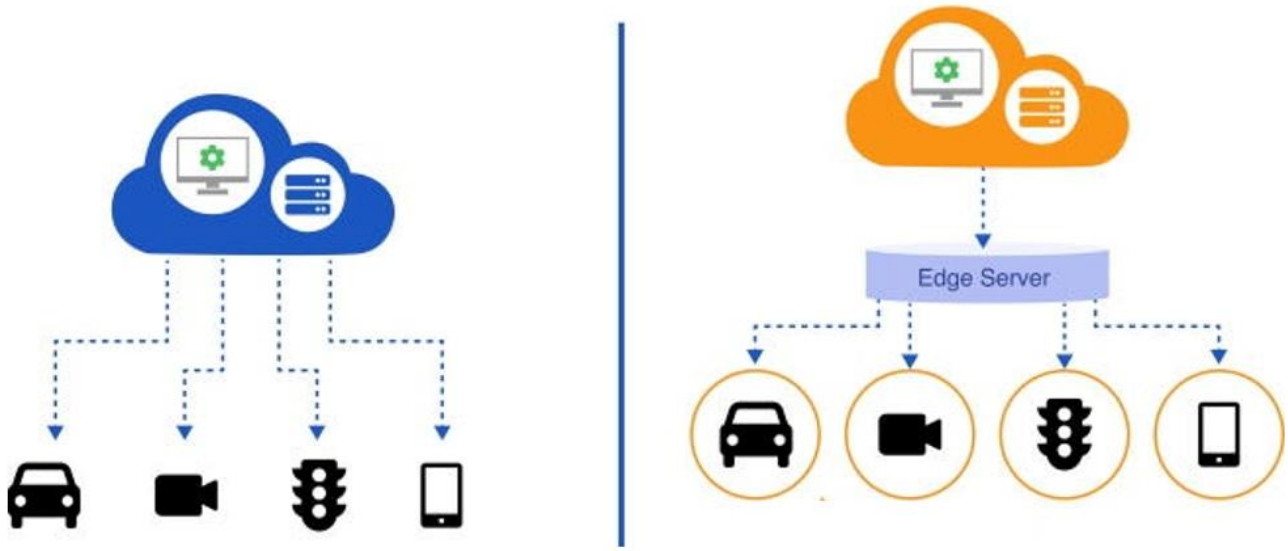


Рисунок 1.1 – Порівняння концепції хмарних обчислень та edge обчислень

Роль шлюзів у edge-обчисленнях виходить за межі простої передачі даних: вони забезпечують стійкість до збоїв, інтеграцію з гібридними хмарами (наприклад, через AWS IoT Greengrass чи Azure Edge) та безпеку на периферії, де фізичний доступ до пристроїв може бути обмеженим, але ризики компрометації високі. У розподілених системах шлюзи виконують функції фільтрації трафіку, шифрування каналів і навіть первинної верифікації цілісності, запобігаючи атакам типу man-in-the-middle чи ін'єкції шкідливого коду на рівні периферії. Для Kubernetes це означає розгортання кластера з master-вузлом на шлюзі (Raspberry Pi як control plane) та worker-вузлами на віддалених edge-пристроях, де поди (контейнери) виконують ізольовані завдання, такі як обробка сенсорних даних чи edge-AI. Така модель дозволяє масштабувати обчислення горизонтально, але ставить нові виклики: обмежені ресурси (CPU, RAM, енергоспоживання),

нестабільність мережі та потреба в апаратній довірі для захисту подів від модифікацій.

У контексті edge-пристроїв на базі Raspberry Pi шлюзи набувають особливого значення як "розумні хаби", що координують гетерогенні мережі сенсорів і ефекторів. Наприклад, у промисловій автоматизації шлюз може керувати подом для моніторингу вібрацій обладнання, використовуючи Kubernetes для динамічного масштабування ресурсів залежно від навантаження, а також для інтеграції з TPM-модулями (наприклад, Infineon OPTIGA SLB 9670 через SPI) для апаратної атестації. Це забезпечує не лише ефективність, але й стійкість: при втраті зв'язку з хмарою шлюз продовжує локальну обробку, зберігаючи дані в буфері та відновлюючи синхронізацію. Загалом, edge-обчислення з акцентом на шлюзи трансформують традиційні централізовані системи в децентралізовані, де Kubernetes грає роль оркестратора, а верифікація подів стає ключовим елементом для запобігання каскадним збоям у критичних інфраструктурах.

1.2 Віддалена атестація в хмарних середовищах

Головною причиною серйозних проблем безпеки в хмарних обчисленнях є обмежений контроль і видимість орендарів над інфраструктурою, яку повністю керує постачальник послуг [11-14]. Зі зростанням популярності хмарних рішень збільшується й обсяг конфіденційних, чутливих та критично важливих для бізнесу даних, що зберігаються на платформах, керованих великими сторонніми організаціями.

З іншого боку, постачальник також змушений довіряти своїй інфраструктурі програмному забезпеченню орендарів, яке може становити потенційну загрозу. Таким чином виникає взаємна проблема встановлення та підтримки довіри між сторонами. Розробка механізмів, які дозволили б орендарю переконатися в безпеці та цілісності своєї системи, принесла б користь не лише йому, а й підвищила б упевненість постачальника в загальній захищеності інфраструктури.

Особливу увагу приділено послугам типу IaaS, оскільки ця модель має найнижчий рівень абстракції, що дає орендарю найбільше можливостей для контролю та моніторингу базової інфраструктури. Саме в IaaS, де орендар несе найбільшу відповідальність за конфігурацію та управління ресурсами, постачальники наразі не надають необхідних інструментів для створення справді надійного середовища для розміщення чутливих даних. Орендарі стикаються з труднощами у перевірці цілісності платформи під час розгортання в хмарі та в забезпеченні її захищеності протягом усього часу виконання обчислень [16].

Крім того, сучасні практики не дозволяють орендарям створювати унікальні, апаратно закріплені ідентифікатори для окремих вузлів, які неможливо підробити. Зазвичай ідентифікація базується виключно на програмних криптографічних рішеннях або на сліпій довірі до постачальника. Наприклад, орендарі часто передають незахищені секрети на свої IaaS-вузли через канали постачальника.

Апаратні модулі довіри, такі як TPM, давно розглядаються як ефективне рішення для встановлення довіри, виявлення змін стану системи, що можуть свідчити про компрометацію, та створення криптографічних ідентифікаторів. Проте впровадження TPM в IaaS-середовищах залишається обмеженим через низку перешкод.

По-перше, стандарт TPM та пов’язані з ним технології є складними для реалізації. По-друге, оскільки TPM діє як криптографічний співпроцесор, а не акселератор, він може створювати значні вузькі місця продуктивності — наприклад, генерація одного цифрового підпису може займати понад 500 мілісекунд. Нарешті, TPM є фізичним пристроєм, тоді як більшість IaaS-послуг базується на віртуалізації, яка навмисно абстрагує віртуальні вузли від апаратного забезпечення. Це ускладнює ситуацію: у найкращому разі доступ до апаратного модуля довіри має лише постачальник, а не орендар.

Бажаними властивостями системи довірених обчислень для IaaS є:

– безпечна ініціалізація: система повинна дозволяти орендарям безпечно встановлювати початковий кореневий секрет на кожен хмарний вузол; цей секрет

зазвичай слугує довгостроковою криптографічною ідентичністю вузла та основою для захисту інших секретів і безпечних сервісів;

– моніторинг цілісності системи: орендарі повинні мати можливість постійно відстежувати цілісність хмарних вузлів під час роботи та реагувати на будь-які відхилення протягом однієї секунди;

– безпечна шаровість (підтримка віртуалізації): система має забезпечувати контрольоване орендарем ініціалізацію та моніторинг цілісності у віртуальних машинах з використанням TPM інфраструктури; цей процес виконується у співпраці з постачальником за принципом найменших привілеїв;

– сумісність: система повинна дозволяти орендарям використовувати апаратно закріплені криптографічні ключі у своєму ПЗ для захисту вже існуючих сервісів, таких як шифрування дисків чи керування конфігураціями;

– масштабованість: система має ефективно масштабуватися для безпечної ініціалізації та моніторингу тисяч IaaS-ресурсів з урахуванням їхньої еластичності – динамічного створення та завершення.

Таким чином, впровадження віддаленої верифікації в хмарних середовищах залишається актуальним викликом, що вимагає подолання технічних бар’єрів та створення механізмів взаємної довіри між орендарем і постачальником.

1.3 Відомі методи та засоби віддаленої верифікації Kubernetes pod’ів для edge-пристроїв

Питання віддаленої верифікації контейнерів у Kubernetes набуло значної уваги в останні роки через швидке поширення контейнеризації в розподілених і периферійних середовищах. Традиційні інструменти атестації, такі як Keylime чи OpenCIT, орієнтовані переважно на перевірку цілісності цілого вузла або віртуальної машини, що виявляється недостатнім для мультиорендарних кластерів, де загроза може локалізуватися в одному поді. У edge-сценаріях ця проблема посилюється обмеженими ресурсами пристроїв, нестабільністю мережі та необхідністю гранулярної перевірки окремих навантажень без значного

навантаження на апаратне забезпечення. Тому сучасні дослідження фокусуються на інтеграції апаратних коренів довіри (TPM) з механізмами контейнеризації та оркестрації, щоб забезпечити динамічну верифікацію подів у runtime [1-20].

Огляд літератури показує, що більшість рішень використовують комбінацію IMA для вимірювання виконуваних файлів, TPM для апаратного захисту агрегатів вимірювань та кастомних компонентів Kubernetes (CRD, контролери, тощо) для автоматизації процесу. Особливістю edge-пристроїв є потреба в легких реалізаціях, сумісних з платформами типу Raspberry Pi чи подібними ARM-системами. Розглянемо детальні пропоновані у літературі рішення.

У роботі [21] запропоновано одну з перших повноцінних архітектур гранулярної віддаленої атестації подів у Kubernetes з використанням TPM 2.0 та IMA. Автор розробив систему, де атестація ініціюється орендарем через підписані запити, обробляється верифікатором на керуючій площині, а докази генеруються агентом на робочому вузлі. Для фільтрації вимірювань конкретного пода використано модифікацію ядра Linux (патч ima-cgpath), що додає cgroup-путь до записів IMA. Реалізація інтегрована через кастомні CRD та контролери, забезпечуючи автоматичну реакцію на недовіряючі поди.

У подальшій роботі [22] авторів на базі тієї ж архітектури досліджено інтеграцію з захищеними контейнерами (Kata Containers), що підвищує ізоляцію подів за рахунок апаратної віртуалізації. Показано зменшення поверхні атаки за рахунок комбінації TPM-атестації з захищеним рантаймом.

У роботі [23] представлено комплексне рішення для віддаленої атестації вузлів у Kubernetes кластерах, спрямоване на забезпечення довіри до edge-інфраструктури у хмарних середовищах. Автори розробили систему KubeTEE, яку інтегрували з існуючим фреймворком Keylime для створення повноцінного рішення атестації на базі TPM 2.0. Ключовою інновацією є розширення стандартного процесу атестації вузлів додатковими механізмами верифікації контейнеризованих застосунків через модифікацію IMA підсистеми Linux ядра. Дослідники реалізували прототип на базі Kubernetes версії 1.23 та провели експериментальну валідацію на кластері з трьох фізичних машин, обладнаних

апаратними модулями довіри, демонструючи можливість автоматичної реєстрації робочих вузлів з генерацією ключів ідентифікації атестації та їх верифікацією через ключі підтвердження. Система забезпечує безперервну атестацію з періодичною перевіркою вимірювань цілісності, що дозволяє виявляти компрометацію вузлів у режимі реального часу. Експериментальні результати показали, що додаткові накладні витрати на впровадження атестації є прийнятними для промислових середовищ: час реєстрації нового робочого вузла становить близько 15 секунд, що включає створення ключа ідентифікації, активацію облікових даних через протокол «запит-відповідь» та валідацію сукупних показників завантаження. Автори детально описали архітектуру системи, яка складається з компонентів верифікатора та реєстратора на рівні площини керування, агента на кожному робочому вузлі та додаткових контролерів для інтеграції з програмним інтерфейсом Kubernetes. Важливим внеском роботи є аналіз безпекових властивостей запропонованого рішення в контексті моделі загроз, де розглядаються сценарії компрометації робочих вузлів зловмисником з доступом до мережі кластера. Однак дослідження має обмеження щодо деталізації атестації – система оперує на рівні цілого вузла, а не окремих подів чи контейнерів, що унеможливлює вибіркове видалення скомпрометованих застосунків без перезавантаження всього вузла. Ця робота закладає фундамент для подальших досліджень у напрямку точної атестації контейнеризованих робочих навантажень у хмарних периферійних обчисленнях, демонструючи можливість інтеграції апаратних механізмів довіри з платформою Kubernetes.

Автори статті [24] пропонують архітектурне рішення на базі «Віртуального Кублета» – легковагого компонента-посередника, який функціонує на більш потужному вузлі кластера та імітує присутність периферійного пристрою як повноцінного робочого вузла в Kubernetes. При цьому фактичне виконання контейнерів відбувається на самому периферійному пристрої через спрощене середовище виконання без необхідності запуску повного стеку компонентів Kubernetes.

Автори розробили розширювану архітектуру на основі постачальників, яка дозволяє адаптувати це рішення до різних типів периферійних пристроїв через модульні інтерфейси, забезпечуючи гнучкість у виборі середовища виконання контейнерів та протоколів зв'язку. Експериментальна перевірка проводилася на різнорідній інфраструктурі, що включала одноплатні комп’ютери та інші пристрої інтернету речей з обмеженими ресурсами (512 МБ оперативної пам'яті, одноядерний процесор), де традиційний компонент керування вузлом споживав би понад 80% доступної пам'яті, що робило б розгортання неможливим.

В іншій роботі [25] автори ідентифікують фундаментальні вразливості сучасних платформ оркестрації контейнерів, зокрема Kubernetes та Docker Swarm, включаючи недостатню ізоляцію між контейнерами на одному вузлі, ризики компрометації через уразливості в образах контейнерів, відсутність деталізованого контролю доступу до ресурсів контейнерів та обмежені можливості атестації стану застосунків під час їх виконання. Дослідники розробили архітектуру фреймворку, яка розширює стандартний Kubernetes механізми додатковими валідаційними етапами: перевіркою підписів образів контейнерів перед розгортанням для гарантування їх провенансу, сканування на відомі CVE вразливості через інтеграцію з базами даних вразливостей, та аналізу в режимі runtime для виявлення аномальної поведінки контейнерів після їх запуску. Прототип системи реалізовано як набір хуків та контролерів, що перехоплюють запити на створення та застосовують набір розроблених безпекових політик перед фактичним плануванням на робочих вузлах.

Також варто згадати технічний звіт NIST SP 800-147 "Рекомендації щодо захисту BIOS", який є фундаментальним документом, що встановлює базові принципи захисту мікропрограмного забезпечення та забезпечення довіреного завантаження систем, що безпосередньо релевантне для розуміння апаратного кореня довіри у контексті віддаленої атестації периферійних пристроїв [26]. Автори роботи систематизують вимоги до захисту BIOS через криптографічну верифікацію оновлень мікропрограм, вимірюваний процес завантаження з використанням TPM для збереження вимірювань у регістрах конфігурації

платформи, та механізми виявлення несанкціонованих модифікацій. Ключовим внеском роботи є визначення трирівневої архітектури довіри: механізм автентифікованого оновлення з цифровими підписами для мікропрограм, безпечний процес завантаження з верифікацією кожного компонента ланцюга завантаження перед передачею управління, та інтеграція з TPM як апаратного кореня довіри для можливостей віддаленої атестації. Документ встановлює, що агрегат завантаження, обчислений з PCR регістрів від нуля до дев'яти, формує криптографічну основу для всіх подальших вимірювань цілісності системи, включаючи записи журналу IMA для застосунків під час виконання. Хоча специфікація розроблена для традиційних систем з BIOS у 2011 році, її принципи були інкорпоровані у сучасні стандарти безпечного завантаження UEFI та TPM версії 2.0, що використовуються у розробленій системі атестації подів на базі Raspberry Pi для валідації довірчого стану робочих вузлів перед їх реєстрацією у кластері Kubernetes.

Таким чином, проблемі віддаленої верифікації Kubernetes-подів для edge-пристроїв приділяється значна увага в сучасних дослідженнях, що відображає актуальність забезпечення безпеки в розподілених контейнеризованих середовищах. Незважаючи на досягнення, усі розглянуті підходи мають суттєві недоліки в контексті edge-пристроїв: більшість вимагає модифікацій ядра Linux або складної конфігурації рантайму, що ускладнює розгортання на стандартних ARM-платформах; атестація вузла призводить до надмірних реакцій (ізоляція всього вузла замість окремого пода); надлишковість продуктивності та трафіку залишається помітним у низькоресурсних середовищах; відсутня повна підтримка апаратних TPM-модулів без віртуалізації. Ці обмеження підкреслюють необхідність розробки легкої, гранулярної системи атестації подів, адаптованої саме до периферійних обчислень з мінімальними змінами базової ОС та оптимізацією для обмежених ресурсів.

1.4 Фреймворк Keylime для віддаленої верифікації

Keylime є відкритим програмним фреймворком для масштабованої віддаленої атестації та моніторингу цілісності систем у хмарних середовищах, розробленим дослідницькою групою з кібербезпеки MIT Lincoln Laboratory та прийнятим як проєкт Cloud Native Computing Foundation [27]. Архітектура Keylime реалізує модель довіри для інфраструктури як послуги, де хмарний провайдер розглядається як напівдовірена сторона, що може бути скомпрометована зловмисниками, однак фізичний доступ до апаратних ресурсів та можливість підробки модулів TPM виключається з моделі загроз.

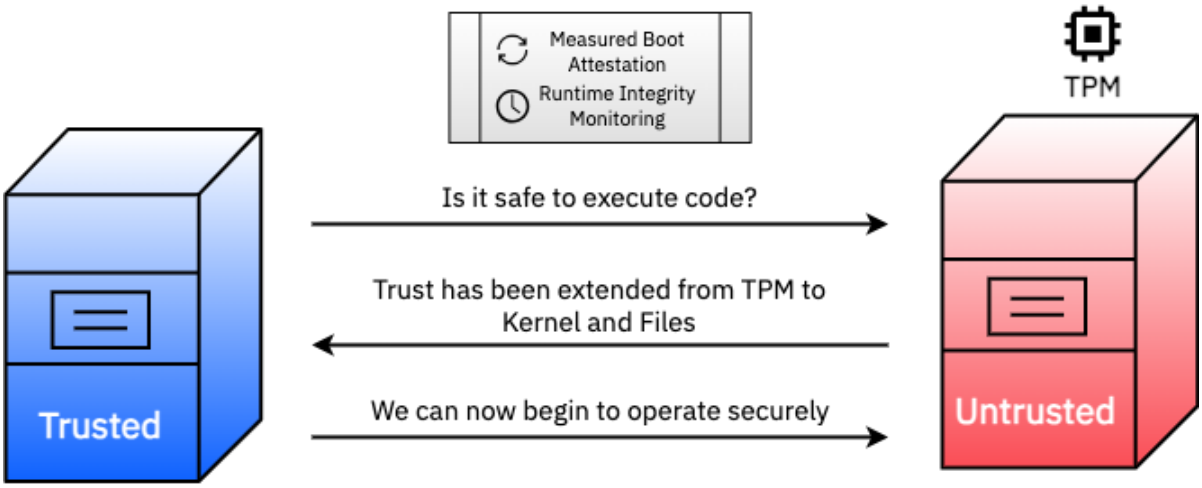


Рисунок 1.1 – Модель верифікації на основі фреймворку Keylime

Фреймворк складається з чотирьох основних компонентів, кожен з яких виконує специфічну роль у процесі атестації. Компонент агенту розгортається безпосередньо на операційній системі вузла, який підлягає верифікації, та відповідає за взаємодію з TPM для генерації криптографічних доказів цілісності системи. Агент керує створенням ключа атестаційної ідентичності, генерує цитати над регістрами конфігурації платформи та збирає журнал вимірювань IMA для передачі верифікатору. Компонент реєстратор виконує функцію реєстрації агентів у системі атестації, отримуючи від кожного вузла унікальний ідентифікатор UUID, публічну частину ключа підтвердження та сертифікат ключа підтвердження TPM.

Registrar здійснює валідацію сертифіката ключа підтвердження через перевірку ланцюга довіри до кореневого сертифіката виробника TPM, після чого виконує критичну процедуру доведення резидентності ключа атестаційної ідентичності всередині того ж TPM, де зберігається ключ підтвердження.

Процедура доведення резидентності реалізується через криптографічний протокол активації облікових даних, заснований на командах TPM MakeCredential та ActivateCredential. Реєстратор генерує тридцятидвобайтовий ефемерний ключ та використовує його для шифрування хешу публічної області ключа атестаційної ідентичності, формуючи зашифровані облікові дані. Сам ефемерний ключ шифрується публічною частиною ключа підтвердження, створюючи виклик активації. Агент отримує обидва зашифровані компоненти та викликає команду TPM ActivateCredential, яка може успішно розшифрувати виклик лише якщо приватна частина ключа підтвердження та ключ атестаційної ідентичності знаходяться в одному TPM. Успішне відновлення ефемерного ключа доводиться через обчислення коду автентифікації повідомлення на основі хешу над ідентифікатором вузла UUID, який агент повертає реєстратору разом з цитатою над регістрами завантаження, підписаною щойно активованим ключем атестаційної ідентичності.

Компонент верифікатор представляє центральний елемент архітектури, відповідальний за проведення фактичної атестації вузлів та обробку отриманих доказів цілісності. Після успішної реєстрації вузла, верифікатор ініціює процес атестації, відправляючи запит до агента з випадковим значенням nonce для забезпечення свіжості відповіді та запобігання атак повторного відтворення. Агент генерує цитату над регістром PCR десять, розширеним підсистемою IMA записами вимірювань виконуваних файлів, підписує її ключем атестаційної ідентичності та повертає верифікатору разом з повним журналом вимірювань IMA. Верифікатор виконує багатоступеневу валідацію отриманих доказів: перевіряє підпис цитати через публічну частину ключа атестаційної ідентичності, отриману з реєстратора, реконструює агрегат IMA шляхом послідовного обчислення операцій розширення над кожним записом журналу та порівнює результат зі значенням PCR десять у

цитаті, після чого кожен запис журналу IMA валідується проти білого списку еталонних значень, що містить хеші довірених виконуваних файлів.

Компонент орендаря надає інструментарій командного рядка для управління вузлами через інтерфейс верифікатора, дозволяючи адміністраторам додавати або видаляти вузли з процесу атестації, перевіряти сертифікати ключів підтвердження проти сховища сертифікатів, отримувати поточний стан довіри вузлів та конфігурувати політики відкликання для автоматичної реакції на виявлення компрометації. Keylime підтримує безперервну атестацію через періодичну генерацію цитат з налаштовуваним інтервалом, типово від декількох секунд до хвилин, що дозволяє виявляти зміни у стані системи з мінімальною затримкою. При виявленні невідповідності між отриманими вимірюваннями та еталонними значеннями, Верифікатор виконує налаштовану політику відкликання, яка може включати видалення вузла з кластера, повідомлення адміністратора або ініціацію автоматичних процедур відновлення.

Архітектура Keylime забезпечує масштабованість через централізовану модель, де один екземпляр верифікатора може обслуговувати тисячі вузлів одночасно, розподіляючи навантаження атестації у часі через набір запитів [28]. Проте ця модель має суттєві обмеження у контексті контейнеризованих середовищ, оскільки Keylime оперує концепцією атестації цілого вузла як єдиної сутності, не розрізняючи окремі контейнери або поди, що виконуються на ньому. Журнал вимірювань IMA містить записи для всіх процесів системи без можливості їх атрибуції до конкретних контейнерів, що унеможливлює гранулярну атестацію індивідуальних застосунків. Як наслідок, при виявленні компрометації будь-якого контейнера на вузлі, єдиною доступною реакцією є видалення або перезавантаження всього вузла, що призводить до припинення роботи всіх інших, потенційно довірених, контейнерів на ньому. Відсутність інтеграції з оркестраційними платформами на рівні об'єктів управління означає, що Keylime не може взаємодіяти з Kubernetes API для оновлення статусу окремих подів або прийняття рішень щодо їх перепланування на інші вузли, вимагаючи розробки

додаткових компонентів інтеграції для забезпечення seamless роботи з cloud-native екосистемою.

1.5 Специфікація опису криптопроцесора TPM 2.0

Криптопроцесор Trusted Platform Module версії 2.0 (TPM 2.0) є апаратним модулем безпеки, розробленим Trusted Computing Group (TCG) для забезпечення кореня довіри в обчислювальних платформах. На відміну від попередньої версії 1.2, TPM 2.0 пропонує гнучкішу архітектуру з підтримкою різних криптографічних алгоритмів (RSA, ECC, SHA-1/SHA-256), ієрархій ключів та політик авторизації, що робить його універсальним для сучасних систем, включаючи віртуалізацію, хмарні та edge-середовища [29].

Основними компонентами TPM 2.0 є регістри конфігурації платформи PCR, які слугують для зберігання агрегованих вимірювань стану системи. PCR "розширюються" послідовними хеш-значеннями: нове значення обчислюється як hash(старе значення АБО нове вимірювання), що створює незворотний ланцюг довіри від моменту завантаження. Ключовою особливістю є Endorsement Key (EK) – унікальна пара ключів (RSA або ECC), створена виробником і записана в TPM назавжди; приватна частина EK ніколи не покидає чіп і використовується для доведення автентичності пристрою через сертифікат.

Для атестації TPM 2.0 генерує Attestation Identity Key (AIK) – обмежену пару ключів, створену всередині модуля, публічна частина якої сертифікується через EK. Процес створення AIK включає challenge-response (MakeCredential/ActivateCredential), що доводить резидентність ключа в конкретному TPM. Важливою функцією є Quote – криптографічний звіт про значення вибраних PCR, підписаний AIK і доповнений nonce для захисту від повторних атак. Цей звіт дозволяє віддаленому верифікатору переконатися в стані платформи, порівнюючи отримані значення з еталонними.

TPM 2.0 підтримує кілька ієрархій ключів, політику авторизації та механізми відновлення. Модуль діє як співпроцесор, виконуючи операції (підпис,

шифрування) всередині захищеного середовища, що унеможливлює витяг приватних ключів. У контексті віртуалізації підтримується vTPM, а для edge-пристроїв – інтеграція через SPI/I2C (наприклад, OPTIGA SLB 9670). Специфікація TPM 2.0 (Library Specification 1.59+ та Platform Architecture) забезпечує сумісність з measured boot та remote attestation, роблячи його основою для систем на кшталт Kubernetes pod attestation, де апаратна довіра комбінується з IMA для гранулярної перевірки контейнерів.

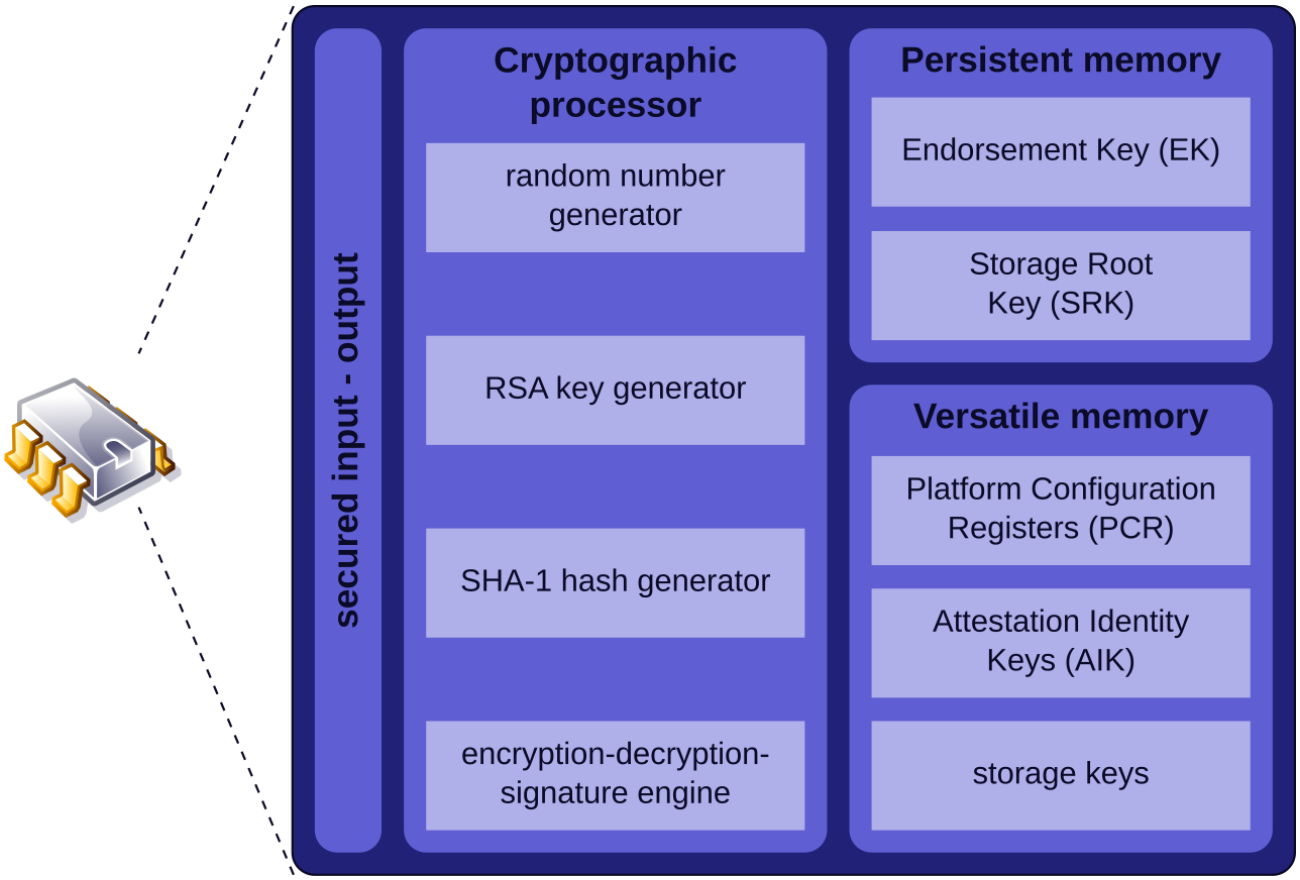


Рисунок 1.3 – Специфікація опису криптопроцесора TPM 2.0 [29]

1.6 Постановка задачі

З метою усуненя недоліків, виявлених у відомих рішеннях, зокрема неможливості гранулярної верифікації окремих Pod'ів у фреймворку Keylime та відсутності нативної інтеграції з Kubernetes API, необхідно розробити спеціалізовану архітектуру віддаленої верифікації, адаптовану до специфіки

контейнеризованих застосунків на edge-пристроях. Такий підхід передбачає створення модульної системи компонентів, інтегрованих безпосередньо в оркестраційну платформу через механізми розширюваності Kubernetes, що дозволить здійснювати атестацію на рівні окремих Pod'ів з можливістю точкового виявлення та усунення скомпрометованих застосунків без впливу на довірені сервіси.

Для вирішення задачі проєктування архітектури та реалізації протоколу віддаленої верифікації Kubernetes Pod’ів для edge-пристроїв на базі Raspberry Pi слід виконати наступні етапи:

1. дослідити процес віддаленої верифікації та пов’язані з ним механізми безпеки;
2. проаналізувати відомі методи, засоби та фреймворки віддаленої верифікації комп’ютерних та вузлів кластерних систем, дослідити їх механізми функціонування, виявити їх недоліки та проаналізувати шляхи вдосконалення;
3. розробити модель процесу віддаленої верифікації Kubernetes подів для edge-пристроїв;
4. розробити архітектуру віддаленої верифікації Kubernetes pod’ів;
5. розробити протокол віддаленої верифікації Kubernetes Pod’ів для edge-пристроїв на базі Raspberry Pi;
6. здійснити розгортання кластеру та провести експериментальні дослідження протоколу віддаленої верифікації kubernetes pod’ів для edge-пристроїв на базі raspberry pi;
7. провести оцінку ефективності пропонованої архітектури та протоколу віддаленої верифікації через порівняння з базовою конфігурацією Kubernetes кластера без механізмів верифікації, а також з реалізацією на базі фреймворку Keylime.

2 МОДЕЛЬ ПРОЦЕСУ ВІДДАЛЕНОЇ ВЕРИФІКАЦІЇ KUBERNETES ПОДІВ

2.1 Процес віддаленої верифікації та пов’язані з ним механізми безпеки

Віддалена верифікація є ключовим механізмом у сфері довірених обчислень, що дозволяє одній стороні (верифікатору) переконатися в цілісності та автентичності стану віддаленої платформи (атестатора) без фізичного доступу до неї. Процес базується на криптографічних вимірюваннях стану системи – обчисленні хешів від прошивки, завантажувача, ядра операційної системи, бібліотек та виконуваних файлів – які захищаються апаратним модулем довіри, таким як Trusted Platform Module (TPM). TPM забезпечує зберігання цих вимірювань у спеціальних регістрах PCR, де значення «розширюються» послідовно: нове хеш-значення комбінується з попереднім, утворюючи незворотний ланцюг довіри від кореня, яким є сам TPM.

Основна ідея верифікації полягає в тому, що верифікатор генерує «доказ»– криптографічний звіт про значення PCR, підписаний приватним ключем AIK, який створюється всередині TPM і не може бути витягнутий. Верифікатор отримує цей доказ разом з журналом вимірювань (наприклад, IMA) і проводить оцінку: перевіряє підпис, перераховує агрегат з логів для порівняння з PCR та зіставляє окремі хеші з еталонними значеннями білого списку. Якщо все збігається, платформа вважається довіреною; інакше – виявляється компрометація, наприклад, модифікація коду чи ін’єкція шкідливого програмного забезпечення.

У контексті контейнеризації та Kubernetes процес верифікації ускладнюється через динамічну природу подів: кожен под є ізольованим процесом із власним набором файлів, бібліотек та залежностей, а кластер може містити тисячі подів на багатьох вузлах. Традиційна атестація вузла перевіряє лише весь вузол, залишаючи сліпу зону для атак на окремі контейнери (наприклад, підміну образу). Тому сучасні підходи прагнуть до гранулярної верифікації на рівні подів, де вимірювання фільтруються за ідентифікаторами контейнера, а докази включають тільки

релевантні записи IMA. Це вимагає інтеграції з контейнерним рантаймом та оркестратором, щоб атестація могла ініціюватися як за запитом орендаря, так і періодично.

Узагальнену схему системи верифікації Kubernetes Pod'ів для edge-інфраструктури подано рис. 2.1.

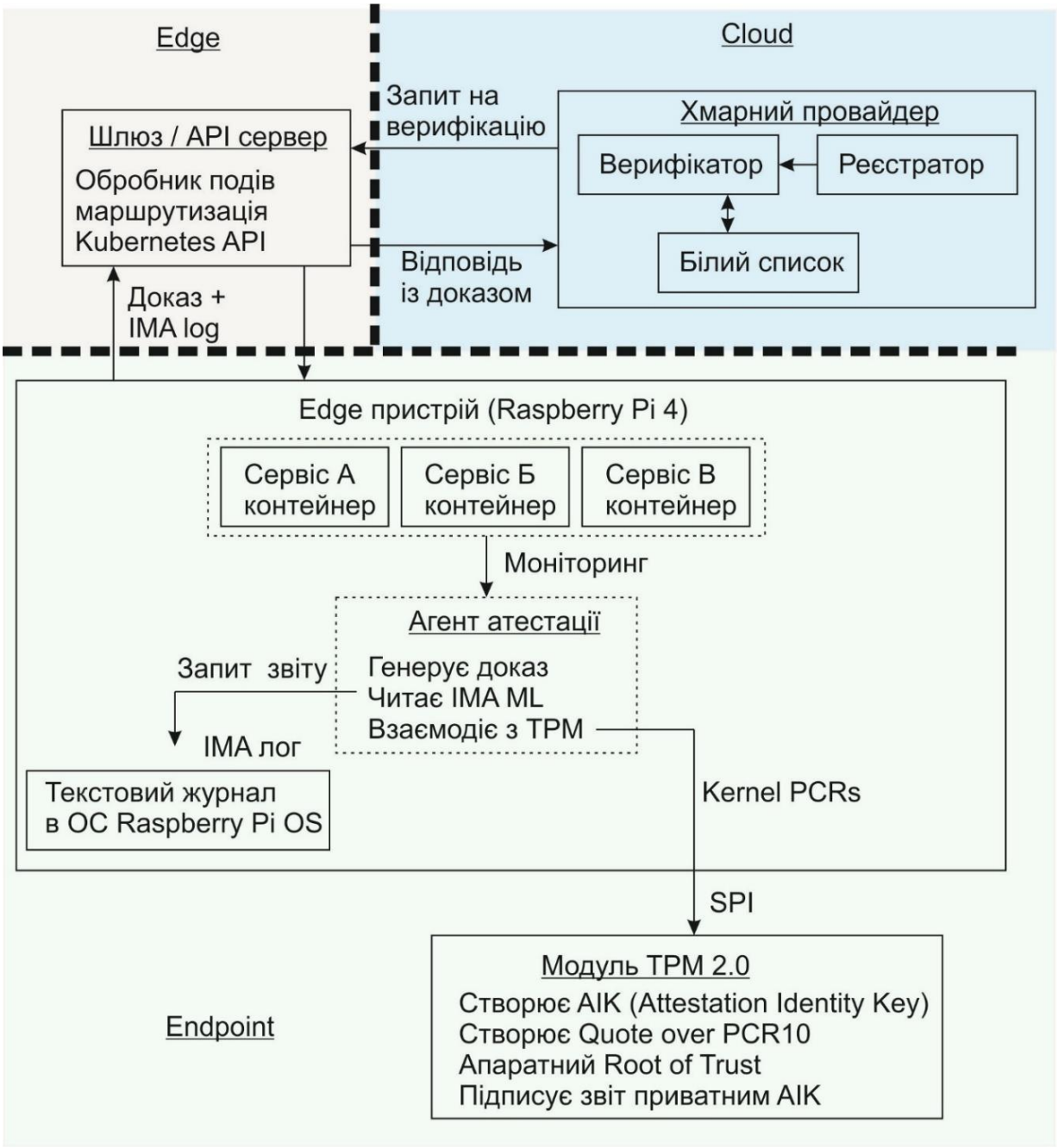


Рисунок 2.1 – Узагальнена схема системи верифікації Kubernetes Pod'ів для edge-інфраструктури

Пов’язаними механізмами є (IMA) у Linux, яка автоматично вимірює файли перед виконанням і записує хеші в журнал, та Appraisal Policy – набір правил для оцінки доказів (наприклад, порівняння з Golden Measurements). У периферійних обчисленнях верифікація набуває додаткового значення через обмежену пропускну здатність і ризик фізичного втручання: апаратний TPM (або віртуальний для тестування) слугує коренем довіри, а протокол повинен бути легким, щоб не навантажувати пристрої типу Raspberry Pi. Загалом, віддалена верифікація перетворює статичну довіру (на етапі розгортання) на динамічну, дозволяючи виявляти загрози в runtime та автоматично реагувати (ізоляція, видалення пода), що є основою архітектур нульової довіри у розподілених системах.

2.2 Модель процесу віддаленої верифікації Kubernetes подів для edge-пристроїв

Запропонована модель процесу віддаленої верифікації Kubernetes-подів є основою для всієї проєктованої архітектури та протоколів у даній роботі, оскільки вона забезпечує формальний і структурований опис взаємодії компонентів, станів довіри та криптографічних операцій, необхідних для реалізації гранулярної атестації в розподілених edge-середовищах. Використання множин для визначення компонентів, ключів, вузлів і подів, а також математичних виразів для умов реєстрації, генерації доказів та оцінки результатів дозволяє чітко моделювати ланцюг довіри від апаратного кореня (TPM на Raspberry Pi) до runtime-виконання окремих контейнерів, що є критичним для обмежених ресурсів edge-пристроїв. Разом із тим така формальна основа робить архітектуру прозорою для аналізу безпеки, спрощує впровадження протоколів (реєстрації вузлів та атестації подів) і полегшує подальшу оптимізацію, наприклад, за рахунок вбудованих механізмів фільтрації IMA namespace ID чи інтеграції з апаратними TPM-модулями типу OPTIGA SLB 9670. У контексті роботи модель слугує не лише теоретичним каркасом, а й практичним керівництвом для реалізації компонентів на k3s,

забезпечуючи баланс між рівнем захисту, гранулярністю перевірки та низьким споживанням ресурсів.

Запропонована модель процесу віддаленої верифікації подів у Kubernetes-кластері для edge-пристроїв формально опишемо як систему із множиною компонентів, станів та переходів, що забезпечують встановлення та підтримку ланцюга довіри від апаратного кореня до виконання у процесі роботи контейнерів. Визначимо базові множини та відношення, які лежать в основі моделі.

Задамо множину компонентів наступним чином:

$$C = \{R, WH, A, ASC, CSC, PH, ARC, V, WLP, PW\}, \quad (2.1)$$

де R – модуль реєстратора, WH – обробник робочих вузлів, A – множина модулів агентів (по одному на кожен робочий вузол), ASC – множина об'єктів агенту стану, CSC – контролер стану кластера, PH – обробник подів, ARC –множина об'єктів CRD, що виконують запит на атестацію, V – верифікатор, WLP – підсистема зберігання білого списку, PW – підсистема моніторингу подів.

Множина вузлів кластера $N = N_m \cup N_w$, де N_m – керуюча площина (master-вузли), N_w – робочі вузли (edge-пристрої на базі Raspberry Pi). Кожен вузол $n \in N_w$ асоційований з одним агентом $a_n \in A$ та одним об'єктом стану $asc_n \in ASC$.

Криптографічні ключі формуються як множина $K = K_{EK} \cup K_{AIK} \cup K_T \cup K_V$, де K_{EK} – множина endorsement keys (заводські ключі TPM), K_{AIK} – множина attestation identity keys (по одному на вузол), K_T –множина ключів орендарів, K_V – ключі верифікатора та спільні секрети.

Стан довіри кластера визначається функцією $T = N_w \cup P \to \{trusted, untrusted\}$, де P – множина всіх подів у кластері. Стан зберігається в ASC, тобто:

$$\forall n \in N_w, \forall p \in Pods(n): T(p) = asc_n.status[p.uid]. \quad (2.2)$$

Процес реєстрації робочого вузла моделюється як послідовність переходів стану вузла $s_n \in \{unregistered, registered, registering, failed\}$. Перехід $s_n \coloneqq refistered$ відбувається лише за умови виконання умов:

$$Reg(n) = \left(EK_{cert_n} \in R.CA\right) \wedge (AIK_n \in R.AIK) \wedge (Quote_{boot}(PCR_{0-9}, AIK_n) \in WLP.boot_{ref}) \wedge (HMAC_{Ke}(UUID_n) = valid), \quad (2.3)$$

де $Quote_{boot}$ – криптографічний звіт про агрегат завантаження, підписаний AIK.

Процес атестації пода p на вузлі n формально опишемо як функцію $Attest(p)$, що повертає булеве значення довіри:

$$Attest(p) = \bigwedge_{m \in IMA_p} (hash(m) \in WLP.ref) \wedge \left(Extend\left(IMA_p\right) = PCR_{10}\right) \wedge \left(Sig_{AIK_n}\left(Quote(PCR_{10}, nonce)\right) = valid\right), \quad (2.4)$$

де $IMA_p \subseteq IMA_{ML_n}$ – підмножина записів журналу IMA, відфільтрована за простором імен ID пода p, $Extend$ – операція розширення PCR значеннями IMA, $nonce \in \mathbb{N}_{32}$ – одноразовий елемент, згенерований верифікатором.

Дана формула описує математичну умову для перевірки (верифікації) довіри до конкретного пода (контейнера) p у системі віддаленої верифікації Kubernetes. Вона визначає, чи под вважається «надійним», базуючись на трьох незалежних, але взаємопов'язаних перевірках, які гарантують цілісність коду, апаратну автентичність та захист від підробок. Якщо всі частини виконуються істинно (тобто "і" між ними), то $Attest(p) = true$, і под проходить атестацію; інакше – $false$, тобто под маркується як под з відсутністю довіри і система реагує відповідним чином, наприклад, видаляє даний под. Проаналізуємо детальніше кожну із трьох складових.

Перша частина є перевіркою на білий список для вимірювань m пов'язаних із подом p.

$$\bigwedge_{m \in IMA_p} (hash(m) \in WLP.ref). \tag{2.5}$$

IMA_p – це підмножина записів з журналу IMA ML, які стосуються саме цього пода (відфільтровані за namespace ID або cgroup-шляхом). Для кожного такого запису m (наприклад, хеш файлу, скрипта чи бібліотеки, що виконується в поді) обчислюється криптографічний хеш (зазвичай SHA-256). Цей хеш повинен точно збігатися з одним із еталонних значень у репозиторії білого списку $WLP.ref$ (що є підсистемою збереження білого списку). Якщо хоч один хеш не збігається (тобто под містить неавторизований файл або модифікований образ контейнера), перевірка провалюється. Це гарантує, що под виконує тільки затверджене, безпечне програмне забезпечення.

У другій частині йде перевірка цілісності ланцюга вимірювань:

$$Extend(IMA_p) = PCR_{10}. \tag{2.6}$$

Функція $Extend$ – це апаратна операція TPM, яка "розширює" регістр PCR 10 послідовними хешами з IMA_p: кожен новий хеш додається до попереднього, утворюючи незмінний агрегат (chain of trust). Обчислене значення повинно точно дорівнювати поточному вмісту PCR 10 у TPM робочого вузла n. Якщо вони не збігаються, це означає, що журнал IMA ML був підроблений або змінений (наприклад, зловмисник спробував вставити фальшиві записи). Ця перевірка доводить, що вимірювання справді відбулися в правильному порядку і без втручання.

Остання складова представляє собою апаратна автентифікація через підпис:

$$(Sig_{AIK_n}(Quote(PCR_{10}, nonce)) = valid). \quad (2.7)$$

Функція $Quote$ генерує криптографічний "звіт" про значення PCR 10, включаючи одноразовий елемент $nonce$ (випадковий номер від верифікатора, щоб уникнути повторних атак). Цей звіт підписується приватним ключем AIK (Attestation Identity Key) TPM вузла n – ключем, який неможливо витягти з чіпа. Верифікатор перевіряє підпис за допомогою публічного AIK, збереженого в реєстрі. Якщо підпис валідний, це доводить, що звіт справді згенеровано цим TPM і не підроблений.

Тоді запит на атестацію моделюється як об’єкт $ark \in ARK$, з атрибутами:

$$ark = \langle p.uid, n, tenant.uuid, HMAC_{secret}(p.uid||n||timestamp)\rangle. \quad (2.8)$$

Даний об'єкт створюється обробником подів після верифікації підпису запиту від орендаря і використовується верифікатором для запуску атестації: спочатку перевіряється HMAC, потім генерується nonce і надсилається агентові. Формула гарантує конфіденційність (UUID не розкриває дані) та цілісність параметрів.

Верифікатор виконує цикл над ARK:

$$\forall ark \in ARK_{new}: (HMAC_{valid}(arc) \wedge p.owner = tenant.uuid) \rightarrow SendRequest(V, a_n, nonce). \quad (2.9)$$

Відповідно до зазначеної вище формули логіка модуля реєстратора полягає у наступному: для кожного нового об'єкта запиту на верифікацію (arc) у множині нових запитів перевіряється дві умови – валідність HMAC (для цілісності даних) та відповідність власника пода ($p.owner$) UUID орендаря ($tenant.uuid$). Якщо обидві істинні ("і"), то верифікатор (V) надсилає запит на атестацію агентові (a_n) на робочому вузлі (n) з одноразовим елементом nonce (для захисту від повторів). Інакше запит ігнорується. Це забезпечує безпечний запуск атестації лише для авторизованих подів.

Відповідно до пропонованої моделі модуль агента генерує доказ атестації:

$$Evidence_p = \langle Quote_{AIK_n}(PCR_{10}, nonce), IMA_p, Sig_{AIK_n}(Quote||IMA_p)\rangle \quad (2.10)$$

Дана формула описує структуру доказу атестації ($Evidence$) для пода p на вузлі n. Вона являє собою кортеж з трьох елементів, згенерований агентом і надісланий верифікатору для перевірки цілісності, зокрема: криптографічний звіт (quote) про значення PCR 10 (агрегат IMA-вимірювань), підписаний AIK вузла з одноразовим nonce для унікальності; відфільтровані записи журналу IMA ML, пов'язані з подом p (хеші файлів, образів контейнерів); підпис AIK над конкатенацією звіту та IMA-логів, що доводить автентичність усього пакета.

Оцінка доказу верифікатором формується відповідно до наступної логічної умови:

$$\begin{aligned} Eval(Evidence_p) \quad (2.11)\\ = \big(Verify_{AIK_n}(Sig) = true\big) \wedge \big(Recompute\big(IMA_p\big) \\ = PCR_{10}\big) \wedge \big(\forall h \in Hashes\big(IMA_p\big): h \in WLP.ref\big). \end{aligned}$$

Дана формула описує оцінку доказу верифікації $Evidence$ для пода p на вузлі n. Вона відображає логічну умову із трьома перевірками, які верифікатор виконує для підтвердження довіри. Якщо всі істинні ("оператор і"), то $Evidence$ валідний. Ці три умови визначають:

− перевірку підпису Sig на $Evidence$ за публічним AIK вузла – доводить автентичність і відсутність підробки;

− перерахунок агрегату з IMA-логів пода IMA_p і порівняння його із PCR 10 – гарантує, що лог не змінений;

− для всіх хешів h з IMA-логів перевірка на відповідність еталонним у білому списку WLP – підтверджує, що файли/образи дозволені.

Тоді у разі успіху оцінки доказу модуль верифікатора оновлює стан:

$$asc_n.status[p.uid] \coloneqq trusted, CSC.react(asc_n) = \emptyset. \quad (2.12)$$

А у разі невдачі:

$$asc_n.status[p.uid] = untrusted, CSC.react(asc_n) \ni \{delete(p), isolate(n)\}. \quad (2.13)$$

Таким чином успішна перевірка модулем верифікатор $Eval(Evidence_p) = true$ означає, що підпис $Evidence$ валідний, агрегат IMA збігається з PCR 10, а всі хеші з білого списку – дозволені; под позначається як довірений, і агент стану оновлюється без дій. Неуспішна перевірка ($Eval = false$) фіксує недовіру (наприклад, підроблений підпис чи невідповідні хеші), призводить до статусу untrusted у об'єкті стану, та передбачає автоматичних коригувальних заходів, як видалення пода чи ізоляція вузла. В рамках даної кваліфікаційної роботи ці механізми не реалізовувались та не досліджувались.

Підсистема моніторингу подів PW постійно відстежує множину P_n подів на вузлі n:

$$PW.observe(n) \rightarrow asc_n.pods \coloneqq P_n. \quad (2.14)$$

Контролер стану кластера реагує на зміну стану наступним чином:

$$CSC.loop(\quad): \forall asc \in ASC: (asc.status.changed) \rightarrow ApplyPolicy(asc). \quad (2.15)$$

Дана формула описує безперервний цикл керування контролера стану кластера (CSC): для кожного об'єкта агенту стану (asc) з множини ASC перевіряється зміна статусу довіри ($asc.status.changed$). Якщо зміна виявлена (наприклад, після атестації пода), контролер застосовує політику ($ApplyPolicy$),

ініціюючи реакцію – видалення недовіряючих подів чи ізоляцію вузлів. Це забезпечує автоматичне збереження безпеки кластера в реальному часі.

Модуль реєстратора підтримує множини:

$$R.tenants = \{\langle uuid_t, name_t, pubkey_t \rangle\},$$
$$R.workers = \{\langle uuid_n, hostname_n, AIK_n \rangle\}, \quad (2.16)$$
$$R.ca = CA_{TPM}.$$

Дана формула описує структуру даних у модулі реєстратора (R). Зокрема ця структура даних складається із: множина орендарів – кортежі з UUID, назвою та публічним ключем для аутентифікації; множина робочих вузлів – кортежі з UUID, ім'ям хоста та AIK для валідування доказів; множина сертифікатів CA TPM – для перевірки ланцюга довіри. Це забезпечує централізоване зберігання ключів і ідентифікаторів для безпечної взаємодії компонентів.

Обробник робочих вузлів виконує функцію реєстрації:

$$WH.register(n) = (n \in N_w^{new}) \Rightarrow ChallengeResponse(n) \wedge (Reg(n) = true) \quad (2.17)$$
$$\Rightarrow R.workers \leftarrow R.workers \cup \{n\}.$$

Обробник подів обмежує розгортання:

$$\mathrm{PH.deploy(p, manifest)} \quad (2.18)$$
$$= (\mathrm{Sig}_{tenant}(\mathrm{manifest}) = \mathrm{valid}) \wedge (namespace(p) \in enabled)$$
$$\Rightarrow Scedule(p).$$

Таким чином запропонована модель забезпечує формальну основу для аналізу безпеки: ланцюг довіри починається від апаратного кореня EK, проходить через AIK, IMA-вимірювання та завершується динамічною оцінкою стану подів у реальному часі. Використання множин та логічних умов дозволяє довести властивості конфіденційності (tenant UUID не розкриває ідентичність), цілісності

(неможливість підміни Evidence без AIK) та доступності (автоматичні коригувальні дії CSC). Адаптація до edge-пристроїв досягається за рахунок легкої досить реалізації компонентів на k3s та використання вбудованих механізмів IMA namespace identification, що мінімізує обчислювальні витрати при збереженні гранулярності атестації на рівні окремих подів.

2.3 Висновки

Таким чином досліджено процес віддаленої верифікації та пов’язані з ним механізми безпеки. Досліджуваний процес базується на криптографічних вимірюваннях стану системи – обчисленні хешів від прошивки, завантажувача, ядра операційної системи, бібліотек та виконуваних файлів – які захищаються апаратним модулем довіри, таким як TPM. TPM забезпечує зберігання цих вимірювань у спеціальних регістрах PCR, де значення «розширюються» послідовно: нове хеш-значення комбінується з попереднім, утворюючи незворотний ланцюг довіри від кореня, яким є сам TPM.

Також запропонована модель процесу віддаленої верифікації Kubernetes-подів, яка становить теоретичну основу розробленої в роботі архітектури та протоколів. Вона забезпечує формальний опис взаємодії компонентів, станів довіри та криптографічних операцій, необхідних для гранулярної атестації в розподілених периферійних середовищах.

3 АРХІТЕКТУРА ВІДДАЛЕНОЇ ВЕРИФІКАЦІЇ KUBERNETES POD'ІВ ДЛЯ EDGE-ПРИСТРОЇВ НА БАЗІ RASPBERRY PI

3.1 Організація розподіленого edge-кластеру Kubernetes на основі одноплатних комп'ютерних систем Raspberry Pi

Фізичне середовище, у межах якого функціонує запропонована система віддаленої верифікації, являє собою розподілений edge-кластер Kubernetes, побудований на основі одноплатних комп'ютерів Raspberry Pi. Такий кластер розгортається на периферії мережі, безпосередньо поблизу джерел даних або керованих фізичних процесів, що дозволяє мінімізувати затримки та зменшити залежність від централізованих хмарних ресурсів. Водночас фізична доступність edge-пристроїв та їх обмежені обчислювальні ресурси формують додаткові вимоги до безпеки та контролю цілісності виконання програмного забезпечення.

Кластер складається з центрального керуючого вузла та множини робочих вузлів, які спільно формують єдине обчислювальне середовище. Керуючий вузол, що може бути реалізований у вигляді окремого сервера або більш потужного edge-шлюзу, виконує функції головного вузла Kubernetes і відповідає за оркестрацію pod'ів, управління станом кластера та координацію процесів віддаленої верифікації. Саме на цьому вузлі зосереджуються компоненти, які аналізують отримані від edge-вузлів докази цілісності, приймають рішення щодо довіреності виконання та забезпечують реалізацію політик безпеки.

Робочі вузли кластера представлені пристроями Raspberry Pi, які виконують роль edge-вузлів Kubernetes і безпосередньо відповідають за запуск контейнеризованих застосунків. На цих вузлах можуть виконуватися pod'и, що реалізують локальну обробку даних, взаємодію з фізичними сенсорами та виконавчими механізмами, попередню аналітику, а також сервіси, чутливі до затримок. Raspberry Pi розглядаються як фізично незахищені або частково довірені платформи, що зумовлює необхідність постійного контролю їх програмного стану.

Кожен edge-вузол оснащується апаратними та програмними механізмами формування довіри, зокрема модулем TPM 2.0 та підсистемою Linux IMA. У

сукупності ці механізми дозволяють edge-вузлам не лише виконувати pod’и, але й виступати активними учасниками процесу віддаленої верифікації, надаючи криптографічно захищені докази цілісності свого стану та виконуваних контейнерів. Таким чином, Raspberry Pi одночасно виконують функції обчислювальних вузлів і верифікаторів у системі.

У запропонованому фізичному середовищі можливе гнучке розмежування ролей між вузлами залежно від сценарію використання. Частина edge-вузлів може бути орієнтована на обробку сенсорних даних або керування фізичними процесами, тоді як інші – на виконання допоміжних сервісів, агрегацію даних або попередню фільтрацію інформації. Центральний вузол при цьому зберігає глобальний огляд стану кластера та забезпечує єдину точку прийняття рішень щодо довіреності виконання застосунків.

Таким чином, фізичне середовище кластера формується як розподілена edge-інфраструктура, у якій поєднуються обмежені за ресурсами, але функціонально автономні Raspberry Pi та централізований керуючий вузол. Така організація дозволяє реалізувати масштабовану та безпечну платформу для виконання Kubernetes pod’ів на периферії мережі з гарантованою можливістю віддаленої верифікації їх цілісності.

3.2 Архітектура віддаленої верифікації Kubernetes pod’ів

Запропонована архітектура спрямована на забезпечення віддаленої верифікації Kubernetes pod’ів у середовищі edge-обчислень, де обмежені ресурси, фізична доступність пристроїв і підвищений ризик компрометації вимагають посиленого контролю цілісності виконання. Архітектура нативно інтегрується в Kubernetes кластер без модифікації його ядра та базується на стандартних механізмах розширюваності платформи, зокрема спеціальні визначення ресурсів та власні контролери, що дозволяє реалізувати функції верифікації як частину керуючої логіки кластера.

У запропонованій моделі edge-пристрої, зокрема одноплатні комп’ютери Raspberry Pi, виконують роль робочих вузлів Kubernetes. Такі пристрої розміщуються безпосередньо на периферії мережі та відповідають за виконання контейнеризованих застосунків у вигляді pod’ів, часто у фізично незахищеному або напівдовіреному середовищі. Саме ця особливість edge-інфраструктури зумовлює необхідність апаратно підкріпленої верифікації стану як самих вузлів, так і запущених на них pod’ів.

Контрольні та верифікаційні компоненти архітектури розміщуються на головному вузлі Kubernetes, який може бути реалізований як окремий сервер або більш потужний edge-шлюз. Вони відповідають за координацію процесів верифікації, зберігання довірених еталонів, перевірку криптографічних доказів та прийняття рішень щодо довіреності pod’ів. Взаємодія між головним вузлом та робочими вузлами здійснюється виключно через Kubernetes API, що забезпечує узгодженість із декларативною моделлю керування та спрощує масштабування системи.

На кожному Raspberry Pi розгортається спеціалізований агент, який виконується з підвищеними привілеями та взаємодіє з апаратними і програмними механізмами довіри пристрою. У межах даної роботи Raspberry Pi розглядається як edge-платформа, здатна виступати верифікатором, використовуючи програмний TPM 2.0 як корінь довіри та підсистему Linux IMA для збору вимірювань виконуваного коду. Це дозволяє формувати криптографічно захищені докази цілісності, що відображають реальний стан pod’ів на edge-вузлі.

Ключовим архітектурним рішенням є представлення станів вузлів, pod’ів і процесів верифікації у вигляді кастомних ресурсів Kubernetes. Такий підхід дозволяє інтегрувати механізм віддаленої верифікації в подієву модель Kubernetes і забезпечує автоматичну реакцію системи на зміни стану edge-вузлів або pod’ів. У результаті верифікація pod’ів стає безперервним процесом, тісно пов’язаним із життєвим циклом застосунків, що особливо важливо для edge-сценаріїв із динамічним навантаженням і частими оновленнями.

Загальну схему роботи пропонованої архітектури віддаленої верифікації kubernetes pod’ів можна подати через наступу послідовність дій:

1) Под запускається на робочому вузлі, що представлений одноплатною комп’ютерною системою Raspberry Pi.

2) Агент на Raspberry Pi:

- читає дані з TPM;
- перевіряє цілісність коду pod’а;

3) Raspberry Pi надсилає доказ до керуючого вузла.

4) На керуючому вузлі:

- перевіряється доказ;
- вирішує: довірений под чи не довірений под.

5) Kubernetes кластер виконує реагування, тобто дозволяє або блокує або зупиняє под.

Таким чином, запропонована архітектура формує наскрізний ланцюг довіри від апаратної платформи Raspberry Pi до окремого Kubernetes pod’а, поєднуючи апаратні механізми безпеки, програмні засоби контролю цілісності та декларативну модель керування Kubernetes. Це робить можливим застосування віддаленої верифікації pod’ів у розподілених периферійних інфраструктурах і підвищує рівень довіри до виконання контейнеризованих застосунків на периферійних пристроях.

Пропонована архітектура віддаленої верифікації kubernetes pod’ів для edge-пристроїв на базі Raspberry Pi складається із наступних компонентів:

- модуля реєстратора;
- обробника робочих вузлів);
- модулів агентів (на кожному робочому вузлі);
- агенту стану;
- контролера стану кластера;
- обробника подів;
- об’єкту CRD, що виконує запит на атестацію, верифікатора;
- підсистеми зберігання білого списку;

– підсистеми моніторингу подів.

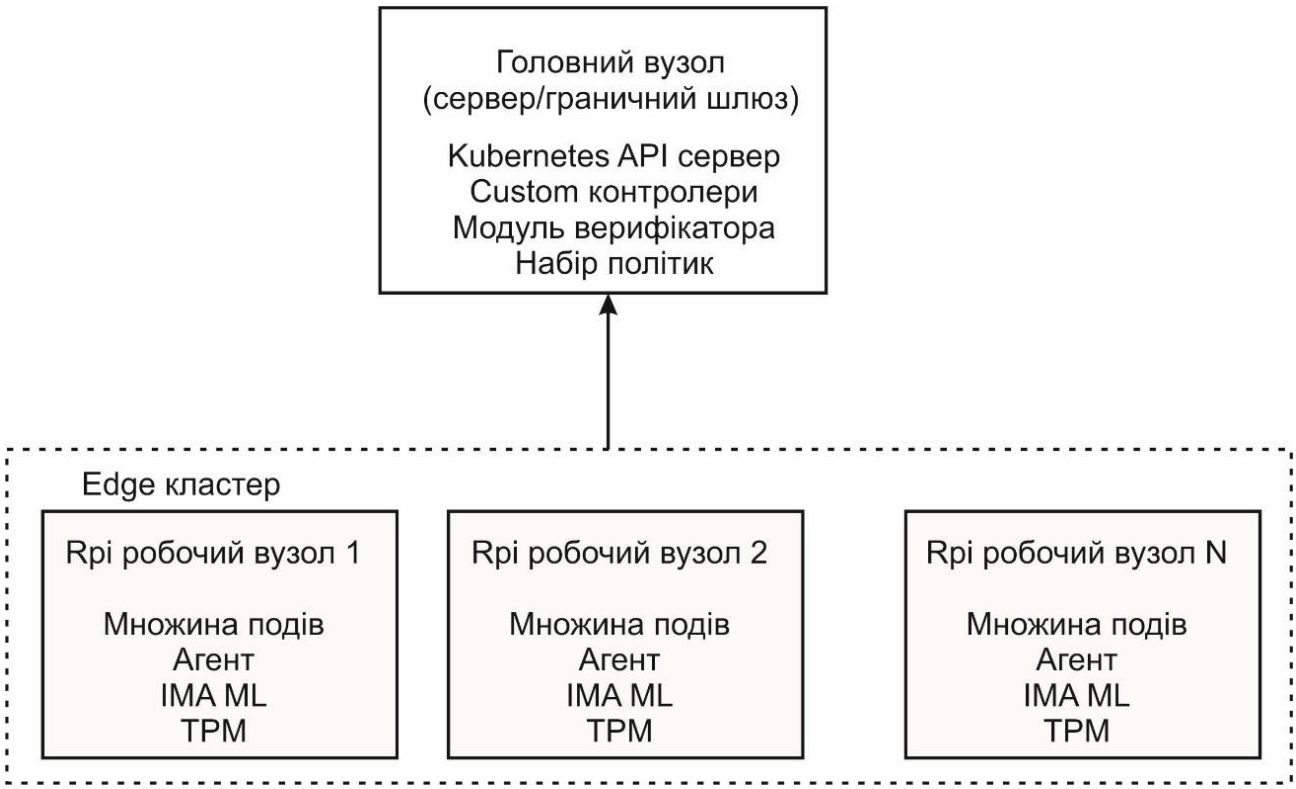


Рисунок 3.1 – Конфігурація Kubernetes edge кластера у якому здійснюється віддаленна верифікація Kubernetes подів

Одним із головних компонентів у пропонованій архітектурі є модуль реєстратора. Модуль реєстратора розгортається на вузлі керуючої площини (master вузлі) і керує асиметричними ключами та сертифікатами, необхідними для забезпечення безпечного виконання системних операцій. Основними функціями реєстратора є:

– реєстрація робочих вузлів у кластері шляхом зберігання їхніх відповідних ключів ідентифікації атестації, що дозволяє валідувати екземпляри доказів, подані модулем агента, розгорнутим на цьому робочому вузлі;

– надання сервісу, який інші компоненти можуть використовувати для взаємодії з ключами орендарів та робочих вузлів для валідування підписів, обчислених за допомогою відповідних приватних частин;

– зберігання інформації про виробників TPM, включаючи проміжні та кореневі сертифікати центрів сертифікації, що дозволяє валідувати сертифікати EK TPM, надані новими робочими вузлами під час їхньої реєстрації;

– ідентифікація залучених орендарів і збереження публічного ключа, який наданий ними під час процесу реєстрації в хмарній інфраструктурі; останній використовується для валідування підписів над запитаними діями, тим самим зберігаючи конфіденційність і запобігаючи несанкціонованому доступу до хмарних ресурсів.

Даний модуль розгортається на керуючій площині, а також керує чотирма ключовими типами даних, забезпечуючи централізований доступ без дублювання компетентностями у інших компонентах.

Спочатку, для орендарів – зареєстрованих користувачів кластера – фіксуються унікальний ідентифікатор UUID (для прив'язки подів до власника), загальна назва та публічний ключ, наданий постачальнику хмари для аутентифікації запитів. Це дозволяє відстежувати поди без розкриття конфіденційної інформації.

Далі, для робочих вузлів – довірених елементів кластера – зберігаються UUID (отриманий під час реєстрації), ім'я хоста та ключ AIK, доведений як належний TPM вузла. AIK слугує для валідування доказів атестації і криптозвітів, гарантуючи апаратну автентичність.

Щодо виробників TPM, модуль реєстратор утримує список визнаних TCG-виробників з комерційними назвами та 4-байтними ідентифікаторами, які повертаються TPM і використовуються в ланцюгу сертифікатів EK для перевірки властивостей чіпа.

Нарешті, для сертифікатів виробників TPM зберігаються кореневі та проміжні CA: з полями загальної назви та самим сертифікатом. Це охоплює ієрархію, де кореневі CA підписують проміжні, а ті – сертифікати для сімейств пристроїв, забезпечуючи повну валідність ланцюга.

Реєстратор надає уніфікований сервіс для інших компонентів: вони надсилають підписи для верифікації, уникаючи самостійного керування ключами

чи перевірки ланцюгів. Такий підхід мінімізує ризики, як-от man-in-the-middle-атаки з підміною ключів, і оптимізує безпеку, зберігаючи всі матеріали в одному захищеному місці.

Ще одним компонентом є обробник робочих вузлів. Обробник робочих вузлів розгортається на головному вузлі і керує процесом реєстрації робочих вузлів, що приєднуються до кластера, завершуючись зберіганням UUID, який унікально ідентифікує кожен робочий вузол і пов'язує його з відповідним AIK у модулі реєстратора. Процес реєстрації робочого вузла включає валідування сертифіката EK TPM, демонстрацію резидентності отриманого AIK та валідування довіреного завантаження.

Реєстрація робочого вузла в системі атестації виконується перед процесом приєднання вузла до кластера, оскільки необхідно створити на ньому агента для взаємодії з його TPM. Після завершення реєстрації обробник робочих вузлів комунікує з модулем реєстратора для зберігання інформації про новододаний вузол і асоціює з ним екземпляр агента стану. Також даний модуль виконує розподіл на новозареєстровані робочі вузли публічний ключ модуля верифікатора, необхідний для аутентифікації запитів атестації.

Іншим компонентом, який автоматично розгортається на кожному робочому вузлі та відповідає за керування взаємодією з TPM, журналом вимірювань IMA та цільовим середовищем атестації є модуль агенту. Основна його функція полягає у створенні AIK ключа та розв'язанні активаційних викликів для останнього під час реєстрації робочого вузла кластера, зборі тверджень, генерації відповідних доказів та поданні цих даних модулю верифікатора для оцінки.

Ще одним компонентом є агент стану – спеціальний ресурс у Kubernetes, який відстежує стан довіри подів орендарів та робочого вузла, на якому вони виконуються. Екземпляр цього ресурсу створюється для кожного робочого вузла для точного асоціювання кожного пода з вузлом, на якому він розгорнутий. Даний модуль діє як централізований реєстр інформації про довіру, оновлюваний виключно модулем верифікатора у відповідь на результат валідування доказів. Варто відзначити, що модифікований стан довіри, асоційований з робочим вузлом

та кожним подом, розгорнутим на ньому, використовується для збереження останнього результату атестації та надання посилання на нього іншим компонентам.

Модуль контролера стану кластера розгортається на головному вузлі і виконує цикл керування для моніторингу стану робочих вузлів та подів, записаних у кожному агенті стану. Основними його функціями є виявлення змін, які можуть вимагати коригувальних дій, таких як видалення пода або вилучення робочого вузла, при умові якщо вони були визначені модулем верифікатора як не довірені. Контролер діє як сторона, що покладається на атестацію, забезпечуючи виконання відповідних дій для збереження цілісності кластера на основі змінного стану довіри робочих вузлів та подів.

Іншим компонентом є обробник подів, що розгортається на головному вузлі й надає орендарям інтерфейс для видачі аутентифікованих запитів на розгортання подів та атестацію. Обробник подів виконує наступні функції:

– верифікує підпис запиту орендаря за допомогою модуля реєстратора для забезпечення його автентичності та незмінності, а потім обробляє його;

– розгортає поди в просторах імен (namespaces), увімкнених для атестації;

– перевіряє, чи под, для якого запитана атестація, належить орендарю, що запитує, а потім видає екземпляр об’єкту CRD, що виконує запит на атестацію для запуску модулем верифікатора процесу атестації з залученим модулем агента.

Об’єкт CRD, що виконує запит на атестацію є спеціальним ресурсом, який визначає всю інформацію, необхідну для запуску атестації над цільовим подом, споживаний модулем верифікатора. Даний об’єкт надає інформацію для взаємодії із модулем агента робочого вузла, на якому виконується под. Також об’єкт CRD, що виконує запит на атестацію включає HMAC, обчислений над параметрами запиту за допомогою секрету, спільного з модулем верифікатора, для запобігання обробки останнім неавтентичних запитів атестації.

Іншим компонентом є модуль верифікатора, який розгортається на головному вузлі та відповідає за проведення атестації подів відповідно до вхідних

екземплярів об’єкту CRD, що виконує запит на атестацію. Модуль верифікатора здійснює:

– виконання циклу керування для моніторингу вхідних запитів атестації;

– використання секрету, що спільний з обробником подів, для валідування отриманого HMAC, тим самим забезпечуючи цілісність запиту атестації;

– запускає процес атестації з Агентом робочого вузла, що хостить цільовий под;

– реалізує політику оцінки для оцінювання отриманих доказів та оновлює атрибути агенту CRD цільового робочого вузла відповідно до результату валідування.

Підсистема зберігання білого списку розгортається на головному вузлі та визначає репозиторій затверджених та довірених еталонних значень для всіх сервісів, конфігурацій робочих вузлів та контейнеризованих застосунків, що виконуються в подах. Основними функціями підсистеми зберігання білого списку є:

– надання модулю верифікатора сервісу для порівняння довірених вимірювань з твердженнями доказів, поданих модулем агентом;

– централізація керування затвердженим програмним забезпеченням та конфігураціями в одному компоненті, забезпечуючи зберігання кожного запису разом з відповідними еталонними вимірюваннями.

Останнім компонентом пропонованої системи є підсистема моніторингу подів, яка також розгортається на головному вузлі й виявляє створення та видалення подів у кластері. Основними функціями підсистеми моніторингу є:

– реалізація циклу керування для моніторингу подів, розгорнутих або видалених з усіх робочих вузлів кластера;

– оновлення відповідного екземпляру агенту стану залученого робочого вузла для відстеження подів, що виконуються та можуть бути предметом атестації.

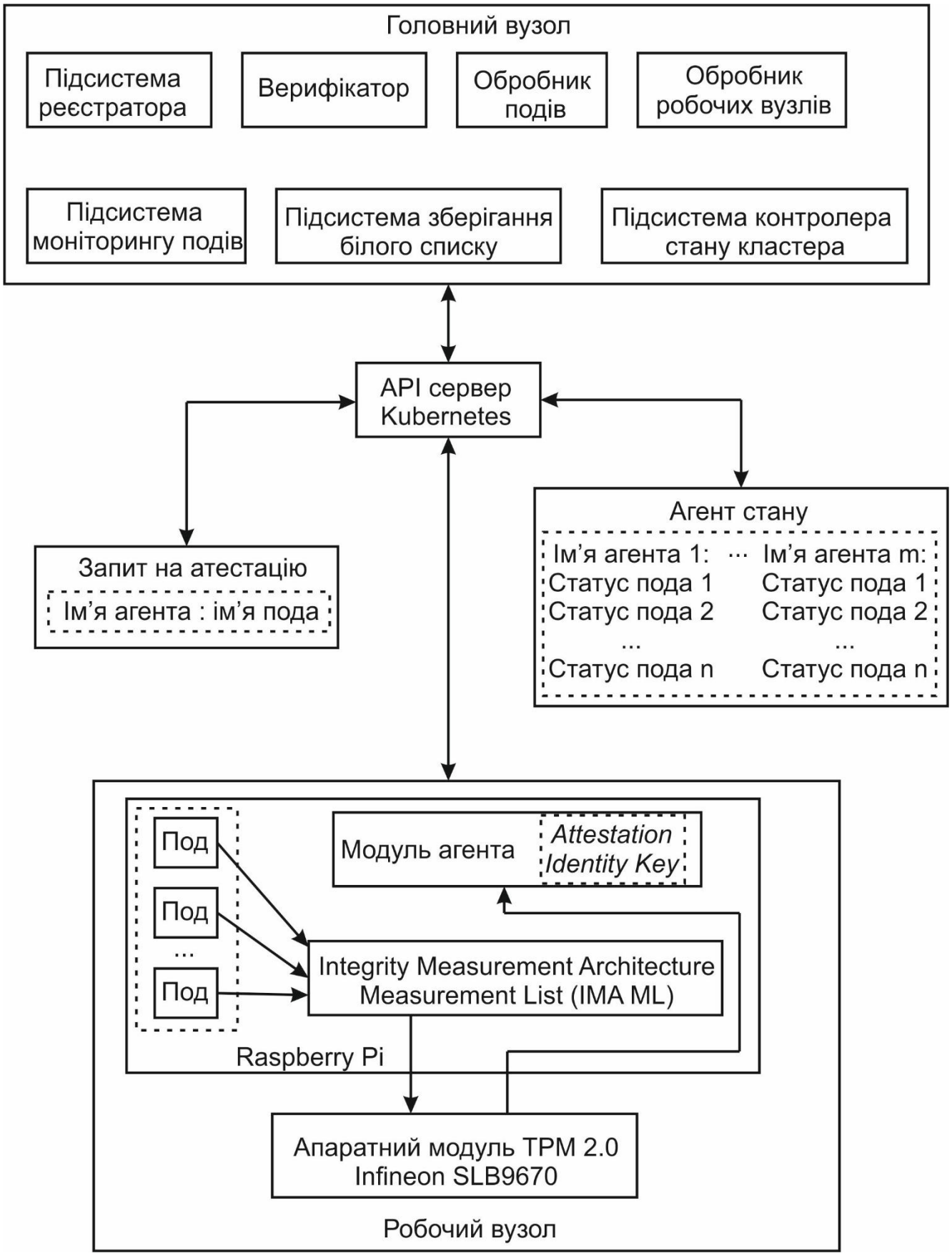


Рисунок 3.2 – Узагальнена архітектура віддаленої верифікації kubernetes pod'ів для edge-пристроїв на базі одноплатних комп'ютерних систем Raspberry Pi

3.3 Модель реєстрації робочих вузлів

Процедура віддаленої верифікації Kubernetes pod’ів для edge-пристроїв на базі Raspberry Pi може розпочатися лише після того, як робочий вузол, що їх хостить, пройде попередню реєстрацію. Цей етап є ключовим для перевірки автентичності вузла та закладання основ довіри, без якої неможливе виконання подальших завдань у системі.

Після успішного завершення реєстрації робочий вузол вважається зареєстрованим, зокрема, його AIK зберігається для подальшого використання. Процес формує ланцюг довіри, починаючи від кореня довіри, наданого TPM вузла, і завершуючи перевіркою AIK на головному вузлі. Ключ AIK у даному випадку слугує основним інструментом для встановлення довіри: він генерується з внутрішнього RoT робочого вузла та передається зовнішнім компонентам, таким як головний вузол та елементи верифікації на ньому.

Процедура реєстрації робочого вузла запускається, коли обробник робочих вузлів, використовуючи цикл моніторингу для вузлів, виявляє нового учасника кластера. Таким чином, реєстрація відбувається поверх стандартного процесу приєднання до Kubernetes-кластера (через kubeadm join), відразу після його успішного завершення.

У процесі реєстрації беруть участь не лише новий робочий вузол та обробник робочих вузлів, а й модуль реєстратора та підсистема зберігання білого списку для перевірки допоміжних даних. Дана процедура реєстрації гарантує, що вузол автентифікований і готовий до участі в верифікації.

Процес реєстрації робочих вузлів включає наступні етапи:

1. Обробник робочих вузлів безперервно відстежує процес ініціалізації робочих вузлів через цикл керування та API Kubernetes. При виявленні нового вузла обробник робочих вузлів виконує розгортання модуля агента на ньому. В свою чергу модуль агента обробляє взаємодію з TPM для надання ідентифікаційних даних обробнику на цьому початковому етапі, а також забезпечує новому вузлу

його екземпляр агента стану. Після цього обробник розпочинає реєстрацію, запитуючи у вузла ідентифікаційну інформацію.

2. Робочий вузол кластера edge пристроїв реагує, збираючи та формуючи такі дані, які надсилаються обробнику робочих вузлів: $UUID$ – унікальний ідентифікатор вузла в модулі реєстратора, AIK_p – публічна область AIK ключа, EK_p – унікальний криптографічний ключ (приватний + публічний), який завод-виробник записує в TPM-чіп один раз назавжди під час виробництва (варто відзначити, що приватна частина EK ніколи не покидає TPM і не може бути витягнута), EK_c – сертифікат EK, безпечно збережений у TPM робочого вузла (він підписаний кореневим або проміжним сертифікатом центру сертифікації) виробника TPM, $hash(AIK_p)$ – ім'я AIK.

3. Обробник робочих вузлів виконує запит до модуля реєстратора, аби перевірити отриманий сертифікат EK за допомогою збережених сертифікатів CA від виробників TPM, підтверджуючи справжність TPM вузла.

4. Отриманий AIK ключ на стороні головного вузла проходить валідацію. Під час цього процесу хеш публічної області AIK ключа перевіряється на відповідність імені AIK, а також публічна область AIK ключа оцінюється на відповідність атрибутам і параметрам стандартного RSA-ключа, придатного для процесу верифікації.

5. Обробник робочих вузлів створює тимчасовий ключ Ke та застосовує його для команди TPM2_MakeCredential(). Ця команда використовується зовні TPM, щоб створити зашифрований «секрет» (credential), який може розшифрувати тільки конкретний TPM модуль, що володіє певним ключем (наприклад, AIK). Таким чином Ke використовується для шифрування імені AIK; результат – зашифрований credential ключа TPM; Ключ Ke шифрується за допомогою EKpub; утворений шифртекст стає викликом активації credential, що дозволяє відправнику підтвердити резидентність AIK у тому ж TPM, де зберігається EK, тим самим закріплюючи довіру до AIK. Врешті зашифрований credential та виклик активації надсилаються робочому вузлу. Таким чином цей процес доводить, що AIK ключ

справді створений у тому ж TPM модулі, де зберігається EK (proof of residency). Це ключовий крок для встановлення довіри до AIK без розкриття приватних ключів.

6. Робочий вузол виконує команду TPM2_ActivateCredential() з отриманими даними, щоб розшифрувати тимчасовий ключ Ke, відновити зашифрований секрет і перевірити його відповідність параметрам AIK. Цей крок надійно прив’язує секрет до ключа AIK. Відновлений ключ Ke використовується для двох цілей: з нього обчислюється HMAC над UUID робочого вузла – це доказ того, що вузол дійсно володіє приватним EK і що AIK створений у тому ж самому TPM (доказ резидентності); перші 8 байтів Ke слугують одноразовим випадковим значенням (nonce) для створення криптографічного звіту (quote) про стан регістрів PCR 0–9. Цей звіт відображає сукупні вимірювання процесу завантаження (boot aggregate). Звіт підписується щойно активованим ключем AIK, якому згодом довірятиме обробник робочих вузлів. HMAC (Keyed-Hash Message Authentication Code – це криптографічний код аутентифікації повідомлення з ключем, тобто HMAC – як підпис з паролем) разом із підписаним звітом (quote) надсилаються Обробнику робочих вузлів.

7. Обробник робочих вузлів перевіряє цілісність отриманого HMAC та валідує криптографічний звіт за еталонними значеннями з підсистеми зберігання білого списку. Валідація HMAC закріплює довіру до AIK, необхідну для перевірки даного криптографічного звіту. Підсистема зберігання білого списку зберігає довірені значення агрегатів завантаження залежно від ОС на робочому вузлі.

8. За умови успішної перевірки агрегату завантаження робочий вузол доводить свою надійність обробнику робочих вузлів, який реєструє його в системі верифікації. UUID, ім'я хоста та AIK_p зберігаються в модулі реєстратора, забезпечуючи їхню стійкість, доки вузол залишається довіреним і частиною Kubernetes-кластера. Таким чином модуль реєстратора підтверджує реєстрацію робочого вузла.

9. Обробник робочих вузлів повідомляє про завершення реєстрації робочому вузлу та передає йому публічний ключ модулю верифікатора, який застосовується

для перевірки та аутентифікації підписаних запитів атестації. Тепер робочий вузол edge кластера готовий до участі в процесах верифікації.

Процес реєстрації робочих вузлів подано на рисунку 3.3.

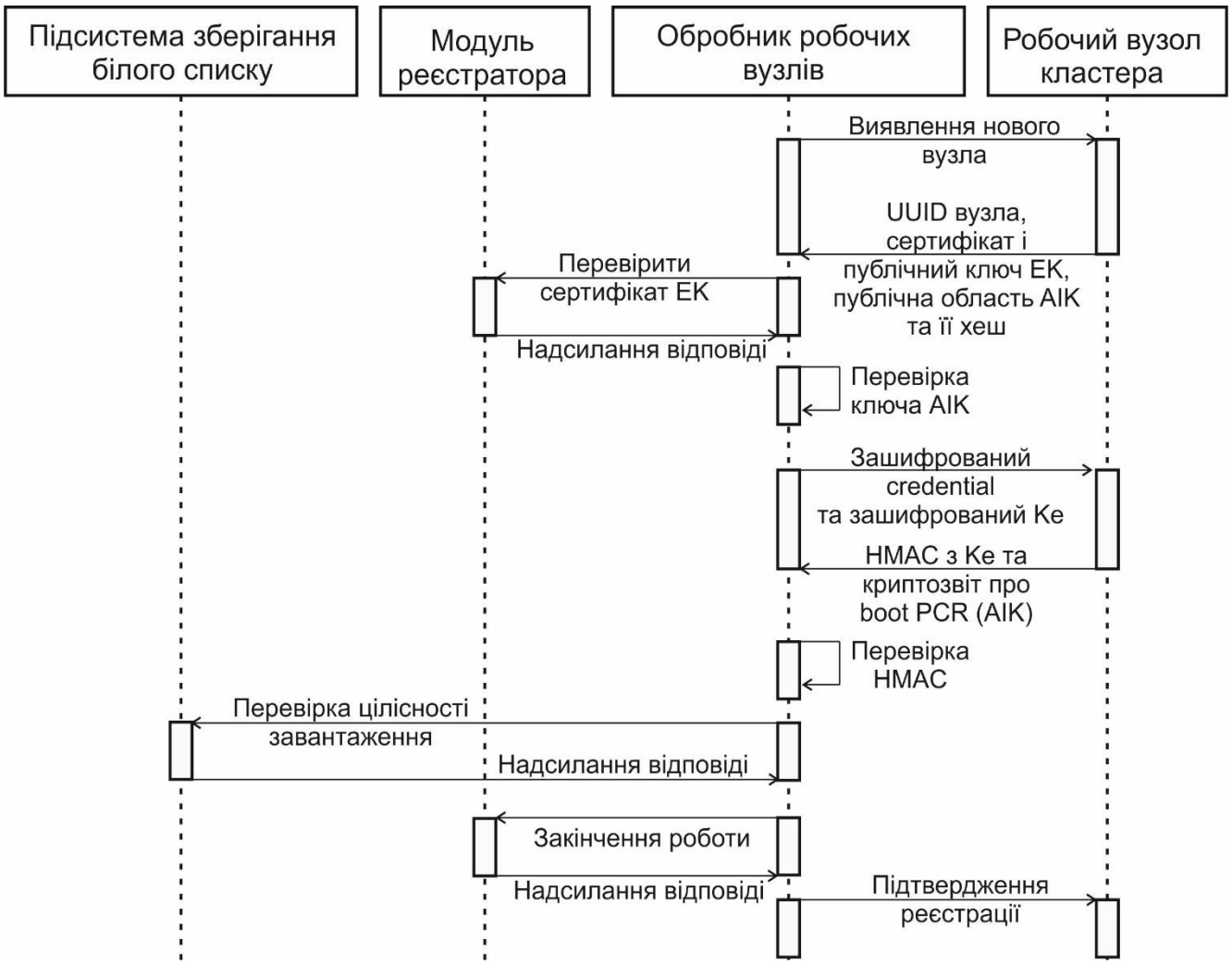


Рисунок 3.3 – Процес реєстрації робочих вузлів

3.4 Процес розгортання подів

Перед тим, як система зможе виконувати віддалену верифікацію подів, необхідно забезпечити надійний механізм перевірки та аутентифікації запитів на створення й розгортання подів. Цю функцію виконує протокол безпечного розгортання подів, який є важливим з кількох причин.

По-перше, він гарантує, що лише орендарі, зареєстровані в системі атестації, мають право надсилати маніфести подів та запитувати їхнє розгортання. Завдяки цьому загальна безпека кластера значно підвищується, оскільки поди можуть створюватися виключно на основі правильно підписаних запитів від авторизованих організацій-орендарів. Приватні ключі орендарів залишаються поза межами кластера або, за наявності TPM на керуючій площині, можуть безпечно генеруватися всередині нього, що мінімізує ризик компрометації через внутрішні атаки та зменшує поверхню можливого впливу.

По-друге, протокол дозволяє пов’язувати унікальний ідентифікатор орендаря з подом уже на етапі його створення, завдяки чому под завжди можна відстежити до свого власника протягом усього життєвого циклу. Цей ідентифікатор генерується без використання будь-якої інформації, що могла б розкрити особу орендаря, таким чином зберігаючи його конфіденційність. Оскільки один і той самий ідентифікатор застосовується до всіх подів, запитуваних одним орендарем, це спрощує виявлення всіх активних подів, що належать йому. Крім того, наявність такого ідентифікатора є критичною для гарантії того, що верифікація проводиться саме над подом, який належить орендарю, що ініціював запит.

Для ідентифікації та аутентифікації орендарів і їхніх запитів на розгортання передбачається, що модуль реєстратор зберігає публічні ключі кожного орендаря, надані ними постачальнику хмарних послуг. Процедура обміну ключами між постачальником і орендарем не деталізується, оскільки її реалізація залежить від обраних механізмів безпеки та внутрішніх політик постачальника. Для узгодженості розглядається варіант позасмугового обміну ключами в рамках укладання угоди про надання послуг.

Процес розгортання подів відбувається наступним чином. Орендар надсилає підписаний маніфест пода до обробника подів. Той, у свою чергу, звертається до Реєстратора для перевірки підпису, оскільки саме модуль реєстратор володіє публічними ключами орендарів і повертає результат верифікації. Якщо підпис дійсний, обробник подів перевіряє, чи простір імен, вказаний у маніфесті, увімкнено для атестації – поди орендарів можуть розгортатися лише в таких

просторах імен. Після успішної перевірки Обробник подів планує створення пода відповідно до всіх параметрів, визначених у маніфесті. Нарешті, підсистеми моніторингу подів виявляє новостворений под і оновлює відповідний екземпляр агента стану на вузлі, де под розгорнуто, що відтепер дозволяє проводити його верифікацію.

3.5 Протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi

У пропонованій системі віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi ключову роль відіграє протокол верифікації подів. Його значення визначається не лише високою чутливістю та критичністю функцій, а й необхідністю залучення всіх компонентів верифікації до спільної роботи. Такий протокол вимагає ретельного проєктування, аби уникнути вразливостей чи непередбачуваних реакцій, оскільки будь-які помилки в його виконанні можуть підірвати надійність верифікаційної системи та загрожувати безпеці всього кластера, який вона захищає.

Детальний огляд взаємодій у протоколі атестації подів розкриває послідовність кроків, що забезпечують послідовну роботу. Спочатку обробник подів отримує підписаний запит на верифікацію від зареєстрованого орендаря, спрямований на один із його розгорнутих подів. Цей запит проходить перевірку на цілісність у модулі реєстратора, де верифікується цифровий підпис, що блокує доступ несанкціонованих осіб до системи верифікації та запобігає витоку конфіденційної інформації про поди в кластері. Далі обробник подів витягує деталі пода з кластера, включаючи робочий вузол, на якому він розміщений – це дозволяє виявити відповідний агент для контакту, а також унікальний ідентифікатор пода в Kubernetes кластері.

Наступним кроком є перевірка, чи под, що верифікується, зафіксований в екземплярі агента стану робочого вузла та пов'язаний з орендарем-запитувачем через його ідентифікатор UUID. Завершуючи підготовку, обробник подів створює

екземпляр об’єкту CRD, що виконує запит на атестацію, до якого додає зібрану інформацію та HMAC, обчислений над цими даними за допомогою спільного секрету з модулем верифікатором.

Модуль верифікатора, у свою чергу, реалізує цикл керування для моніторингу ресурсів типу запит на верифікацію. Він фіксує новий екземпляр CRD, аутентифікує його через перевірку HMAC і генерує одноразовий елемент, аби уникнути повторних атак та гарантувати унікальність процесу атестації. Потім створюється та підписується об’єкт запиту на атестацію на основі даних CRD, який надсилається цільовому агентові, що розташований на робочому вузлі.

Агент, отримавши запит від модуля верифікатора, розпочинає обробку: спочатку валідує підпис, аби переконатися в походженні від надійного джерела, а потім використовує nonce для формування унікального звіту (quote) над PCR 10 — регістром, розширеним IMA для вимірювань користувацьких застосунків. Цей звіт підписується зареєстрованим AIK робочого вузла. Далі агент витягує журнал IMA ML і комбінує всі елементи в доказ атестації, який також підписується AIK перед поверненням модулю верифікатора.

На етапі оцінки модуль верифікатора аналізує доказ атестації: перевіряє підпис Evidence за допомогою AIK, аби виключити фальсифікацію чи втручання інших агентів. Журнал IMA ML розбирається на окремі записи, де оцінюється їхня цілісність; витягаються записи, пов’язані з атестованим подом та контейнерним рантаймом робочого вузла, і формуються в окрему структуру. Обчислюється повний агрегат IMA та порівнюється з значенням PCR 10 із звіту. Нарешті, витягнуті записи з IMA зіставляються з еталонними значеннями з Постачальника білого списку: валідуються виконані файли пода, образ контейнера, з якого він створений, та залежності робочого вузла від рантайму.

У разі успішного завершення всіх етапів под вважається надійним, і процес верифікації завершується позитивно. В іншому випадку, при будь-якій невдачі робочий вузол чи конкретний под позначаються як под, з відсутністю довіри, залежно від характеру проблеми. Модуль верифікатора оновлює екземпляр стану робочого вузла, відображаючи результат атестації, а контролер стану кластера,

помітивши зміну, застосовує свою політику оцінки результатів верифікації та запускає коригувальні заходи для збереження безпеки кластера.

Можливість ініціювати атестацію подів ззовні кластера ідеально вписується в модель взаємодії з хмарними сервісами, підвищуючи гнучкість і зменшуючи складність, яка виникла б від необхідності реалізувати протоколи ідентифікації орендарів на кожному вузлі. Натомість фокус зосереджується на ідентифікації та перевірці цілісності самих вузлів. Такий підхід частково повертає контроль і видимість над інфраструктурою орендарям, що є особливо актуальним у публічних хмарних середовищах, де інфраструктура повністю керується третіми сторонами. Схему комунікації між головним вузлом та робочим вузлом відповідно до протоколу віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi наведено на рис. 3.4.

Таким чином протокол віддаленої верифікації подів складається з п'яти ключових етапів, що забезпечують безпечну перевірку цілісності контейнерів у Kubernetes-кластері з використанням TPM та IMA, зокрема:

1) Ініціація запиту. Обробник подів приймає підписаний запит на атестацію від зареєстрованого орендаря, спрямований на один із його подів. Модуль реєстратора перевіряє цілісність запиту через верифікацію цифрового підпису, блокуючи несанкціонований доступ та витік даних про поди. Обробник подів витягує деталі пода з кластера (вузол, агент, Kubernetes UID), підтверджує зв'язок з орендарем у агенті стану та генерує екземпляр об'єкту CRD, що виконує запит на атестацію з HMAC для захисту даних.

2) Обробка модулем верифікатора. Верифікатор моніторить об'єкти CRD, що виконують запит на атестацію, у своєму циклі керування. Він виявляє новий екземпляр такого ресурсу, аутентифікує його перевіркою HMAC і генерує одноразовий елемент для захисту від повторних атак та гарантії унікальності процесу атестації. На підставі даних з об'єкта запит на верифікацію формується та підписується об'єкт запиту, який надсилається модулю агента на відповідному робочому вузлі.

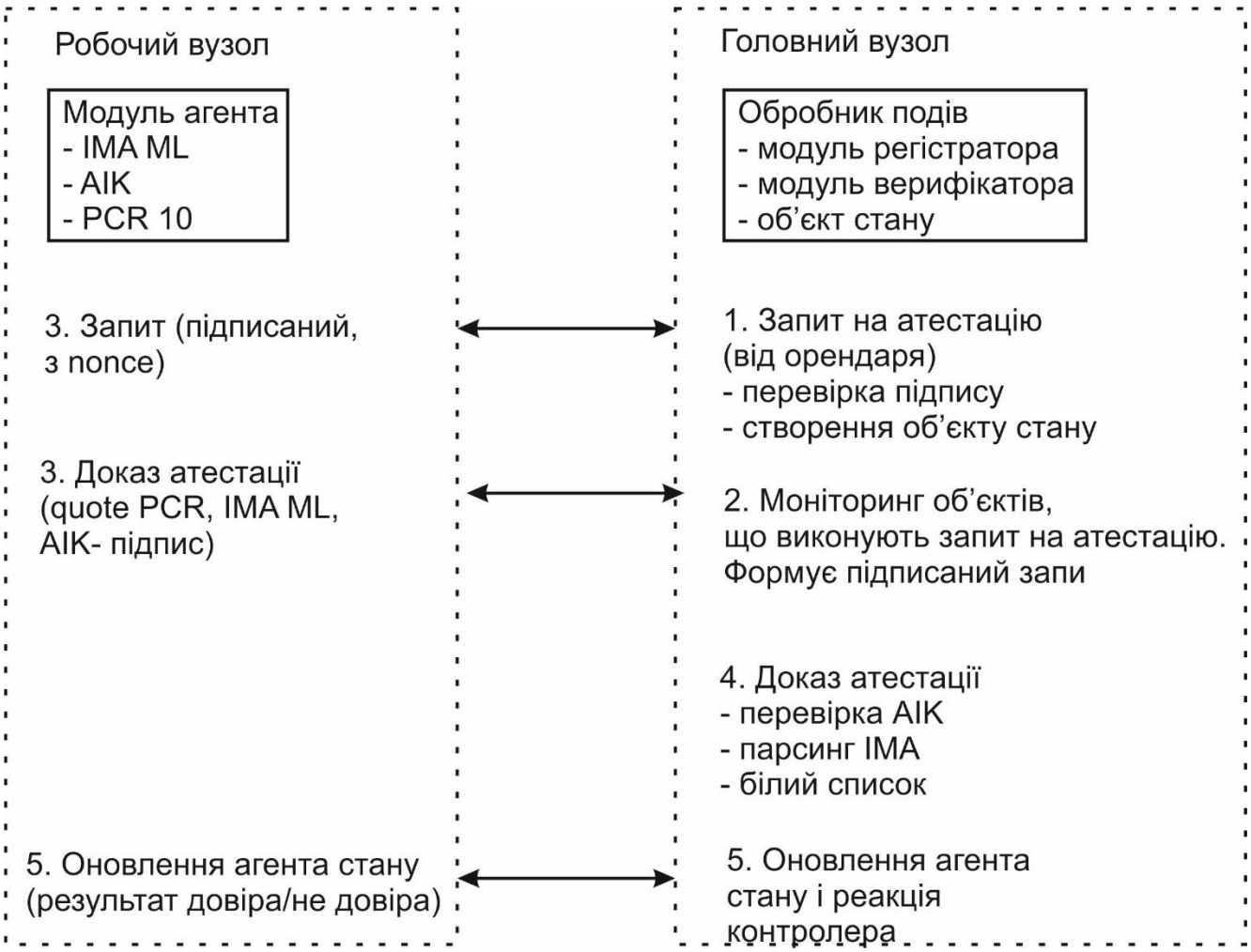


Рисунок 3.4 – Схема комунікації між головним вузлом та робочим вузлом відповідно до протоколу віддаленої верифікації Kubernetes Pod’ів для edge-пристроїв на базі Raspberry Pi

3) Формування доказів модулем агенту. Агент отримує запит, валідує підпис для підтвердження походження від модуля верифікатора, застосовує одноразовий елемент для генерації унікального криптографічного звіту над PCR 10 (розширеним IMA для вимірювань застосунків), витягує журнал IMA ML, об'єднує елементи в доказ атестації та підписує його зареєстрованим AIK, повертаючи модулю верифікатору.

4) Аналіз доказів. Модуль верифікатора перевіряє підпис Evidence за ключем AIK, щоб виключити фальсифікацію, виконує парсинг IMA ML для оцінки записів і витягує дані пода та рантайму (використовуючи ідентифікатор namespace для точної фільтрації), перераховує агрегат IMA для зіставлення з PCR 10 та порівнює

хеші з еталонними значеннями підсистеми збереження білого списку (файли, образи контейнерів, залежності вузла).

5) Фіналізація та коригування. У випадку успіху под маркується як надійний, тоді як при помилці – як под, з відсутністю довіри. Модуль верифікатора оновлює агент стану із результатом, а контролер стану кластера реагує за політикою оцінки, застосовуючи заходи (видалення, ізоляцію) для підтримки безпеки кластера.

3.5 Висновки

Досліджено організацію розподіленого edge-кластеру Kubernetes на основі одноплатних комп'ютерних систем Raspberry Pi, де визначено фізичне середовище функціонування системи верифікації як розподілену edge-інфраструктуру з центральним керуючим вузлом та множиною робочих вузлів на базі Raspberry Pi. Обґрунтовано необхідність апаратних механізмів формування довіри через модуль TPM 2.0 та підсистему IMA для забезпечення криптографічно захищених доказів цілісності стану edge-вузлів та виконуваних контейнерів.

Запропоновано архітектуру віддаленої верифікації Kubernetes Pod'ів, яка нативно інтегрується в Kubernetes кластер через механізми розширюваності платформи без модифікації її ядра. Визначено десять ключових компонентів системи: модуль реєстратора, обробник робочих вузлів, модулі агентів, агент стану CRD, контролер стану кластера, обробник подів, об'єкт CRD запиту на атестацію, модуль верифікатора, підсистема зберігання білого списку та підсистема моніторингу подів. Описано функціональні обов'язки кожного компонента та їхню взаємодію через Kubernetes API, що забезпечує декларативну модель керування та спрощує масштабування системи.

Також формалізовано модель реєстрації робочих вузлів, яка встановлює довірчий ланцюг від апаратного кореня TPM до керуючої площини кластера. Описано дев'ятиетапний протокол реєстрації, що включає виявлення нового вузла обробником робочих вузлів, збір ідентифікаційних даних з TPM, валідацію сертифікату EK через ланцюг довіри до виробника, перевірку резидентності AIK

через криптографічний протокол активації облікових даних TPM2_MakeCredential та TPM2_ActivateCredential, валідацію boot aggregate проти еталонних значень білого списку та фінальне збереження AIK у модулі реєстратора для подальшої верифікації доказів атестації.

Розроблено протокол віддаленої верифікації Kubernetes Pod'ів, що складається з п'яти етапів: ініціації запиту через обробник подів з верифікацією підпису орендаря та створення об'єкту CRD запиту на верифікацію з HMAC, обробки модулем верифікатора через моніторинг CRD ресурсів та генерації одноразового елемента для захисту від повторних атак, формування доказів модулем агента з генерацією криптографічного звіту над PCR 10 та витяганням журналу IMA ML, аналізу доказів модулем верифікатора з перевіркою підпису AIK та зіставленням хешів з еталонними значеннями білого списку, фіналізації з оновленням агента стану та застосування коригувальних заходів контролером стану кластера. Протокол забезпечує наскрізний ланцюг довіри від апаратної платформи Raspberry Pi до окремого Kubernetes Pod'a з можливістю гранулярної верифікації та точкового усунення скомпрометованих застосунків.

4 РОЗГОРТАННЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ПРОТОКОЛУ ВІДДАЛЕНОЇ ВЕРИФІКАЦІЇ KUBERNETES POD'ІВ ДЛЯ EDGE-ПРИСТРОЇВ НА БАЗІ RASPBERRY PI

4.1 Розгортання кластеру та експериментальні дослідження протоколу віддаленої верифікації kubernetes pod'ів для edge-пристроїв на базі raspberry pi

Практична реалізація системи віддаленої атестації для Kubernetes кластера на базі edge-пристроїв Raspberry Pi вимагала комплексного підходу до адаптації існуючих рішень під специфіку обмежених обчислювальних ресурсів. Базовою архітектурою послужила концепція, описана у роботі [1], однак була здійснена модифікація для роботи з апаратним забезпеченням, зокрема одноплатною комп'ютерною системою Raspberry Pi 4 Model B та обладнаним TPM 2.0 модулем Infineon OPTIGA SLB 9670.

Рисунок 4.1 – Робочий вузол Raspberry Pi Model 3 із підключеним криптографічним модулем Infineon OPTIGA SLB 9670

Модуль Infineon OPTIGA SLB 9670 підключався через інтерфейс SPI, що є стандартним для даного чіпа і дозволяє інтеграцію з GPIO Raspberry Pi без додаткових конвертерів.

Підключення здійснювалося за допомогою спеціальної плати-адаптера яка має 2x5-піновий header і просто вставляється у GPIO Raspberry Pi, займаючи піни з 17 по 26. Основні з'єднання включали: живлення 3.3V (пін 1 або 17), землю GND (пін 6 або 9), MOSI (пін 19 або GPIO 10), MISO (пін 21 або GPIO 9), SCLK (пін 23 або GPIO 11) та чіп-селект CS (зазвичай CE0 на піні 24 або GPIO 8, у деяких платах можливе перемикання на CE1). Піни RST (reset) та PIRQ (interrupt) часто залишалися не підключеними, оскільки плата має вбудовану схему скидання. Такий прямий монтаж забезпечував стабільну роботу на частоті до 32 МГц, а після увімкнення SPI в raspi-config модуль розпізнавався системою як /dev/tpm0, дозволяючи повний доступ до функцій TPM для реєстрації та атестації.

Розгортання експериментального стенду розпочалося з підготовки базової операційної системи. На головному вузлі було встановлено Ubuntu Server 22.04.3 LTS з ядром версії 5.15.0-91-generic, оскільки ця конфігурація забезпечувала найкращу сумісність з Kubernetes v1.34 та підтримку необхідних kernel features для IMA. Worker вузол отримав Raspberry Pi OS Lite (Debian Bookworm) з модифікованим ядром 6.1.0, до якого було застосовано патч для підтримки шаблону ima-cgpath.

4.1.1 Модифікація та перезбірка ядра Linux для Pod-орієнтованих вимірювань IMA

Процес модифікації ядра Linux для робочого-вузла Kubernetes розглядався як необхідна умова коректної роботи механізмів вимірювання цілісності в контейнеризованому середовищі. У стандартній реалізації IMA всі вимірювання формуються на рівні вузла і не містять контексту контейнера або Pod’а, у межах якого виконується процес. За відсутності такого контексту неможливо однозначно співвіднести зафіксовані вимірювання з конкретними Pod’ами, що унеможливлює подальшу атестацію та аналіз безпеки на рівні контейнерів у Kubernetes-кластері.

Для усунення цього обмеження було реалізовано модифікацію ядра Linux, спрямовану на розширення структури вимірювань IMA додатковим атрибутом, який відображає приналежність процесу до конкретної контрольної групи. Оскільки в Kubernetes кожен Pod та контейнер ієрархічно представлений у вигляді cgroup, повний шлях cgroup у файловій ієрархії ядра є надійним ідентифікатором виконуваного Pod'а. Саме тому до шаблону вимірювань IMA було додано нове поле cgroup-path, яке зберігає цей шлях у текстовому вигляді.

Технічно модифікація була реалізована шляхом внесення змін до файлу security/integrity/ima/ima_template.c, який відповідає за формування та відображення полів шаблону IMA:

```
static void ima_show_template_data_cgroup(struct seq_file *m,
                                          enum ima_show_type show,
                                          struct ima_field_data *field_data)
{
    struct task_struct *task;
    char *cgroup_path = NULL;

    task = current;
    if (task && task->cgroups) {
        cgroup_path = kmalloc(PATH_MAX, GFP_KERNEL);
        if (cgroup_path) {
            cgroup_path_ns(task_cgroup(task, 0),
                           &init_cgroup_ns,
                           cgroup_path, PATH_MAX);
            ima_show_template_field_data(m, show,
                                         DATA_FMT_STRING,
                                         field_data);
            kfree(cgroup_path);
            return;
        }
```

```
    }
    ima_show_template_field_data(m, show, DATA_FMT_STRING,
                                 field_data);
}
```

У межах цього файлу було додано окрему функцію, призначену для отримання шляху контрольної групи поточного процесу на момент формування вимірювання. Функція використовує покажчик current для доступу до структури task_struct активного процесу, після чого перевіряє наявність асоційованої структури cgroups. За умови її наявності динамічно виділяється буфер пам’яті, у який за допомогою функції cgroup_path_ns формується повний шлях контрольної групи процесу в просторі імен init_cgroup_ns. Отриманий шлях передається механізму виводу шаблонних полів IMA як рядкове значення та включається до відповідного запису вимірювання. У разі помилки виділення пам’яті або відсутності cgroup функція коректно завершує виконання, забезпечуючи стабільність підсистеми IMA та сумісність із наявною логікою формування вимірювань.

Даний код не компілюється як окремий модуль, а є складовою підсистеми безпеки ядра Linux. Після внесення змін ядро перезбиралось стандартним способом із використанням відповідної конфігурації, у якій увімкнено підтримку IMA та контрольних груп, зокрема cgroup v2, що є типовим для сучасних Kubernetes-оточень. У процесі компіляції змінений файл ima_template.c транслюється у відповідний об’єктний файл та лінкується безпосередньо до образу ядра. Отриманий бінарний образ ядра встановлюється на Worker-вузли, після чого вузли перезавантажуються для активації змін.

У результаті застосування патчу механізм IMA починав фіксувати вимірювання з урахуванням cgroup-контексту кожного процесу. Записи в журналах IMA містили не лише хеші та метадані виконуваних файлів, але й повний шлях контрольної групи, який однозначно відображає Pod та контейнер у Kubernetes-кластері. Це дозволило здійснювати верифікацію та аналіз цілісності програмного забезпечення не на рівні окремого вузла, а на рівні Pod’ів, що суттєво підвищувало

точність і практичну придатність механізмів довіреного виконання в хмарних та edge-орієнтованих середовищах.

Перезбірка ядра з урахуванням внесеного патчу виконувалась як завершальний та обов'язковий етап інтеграції змін у робоче середовище робочого вузла. Після модифікації вихідного коду підсистеми IMA було використано офіційне дерево вихідних кодів ядра, що відповідає версії ядра, застосовуваній у дистрибутиві Raspberry Pi OS Lite (https://github.com/RPi-Distro/pi-gen). Це забезпечувало сумісність із завантажувачем, firmware та апаратними особливостями платформи Raspberry Pi, а також відповідність пакетній інфраструктурі дистрибутиву.

Процес перезбірки розпочинався з підготовки конфігурації ядра. За основу бралася стандартна конфігурація Raspberry Pi, яка постачається разом із вихідним кодом, після чого вона доповнювалась параметрами, необхідними для коректної роботи IMA та контрольних груп. У конфігурації ядра було увімкнено підтримку IMA, механізмів безпеки ядра, файлової системи securityfs, а також повну підтримку cgroup та cgroup v2, що є критично важливим для контейнеризованих середовищ Kubernetes. Таким чином, ще на етапі конфігурації гарантувалося, що додане поле cgroup-path матиме коректне джерело даних під час виконання.

Після підготовки конфігурації виконувалась безпосередня компіляція ядра. Залежно від обчислювальних ресурсів збірка могла виконуватися або нативно на самій одноплатній комп'ютерній системі Raspberry Pi, або шляхом крос-компіляції на окремій системі з подальшим перенесенням результатів. У процесі збірки змінений файл ima_template.c компілювався разом з іншими компонентами підсистеми безпеки, після чого всі об'єктні файли лінкувалися у єдиний бінарний образ ядра. Паралельно збиралися модулі ядра, необхідні для роботи драйверів та підсистем, що використовуються у робочому вузлі.

Після успішної компіляції отриманий образ ядра та відповідні модулі встановлювалися у файлову систему цільового вузла. Образ ядра розміщувався у розділі завантаження, який використовується прошивкою Raspberry Pi, а модулі встановлювалися у стандартну ієрархію системних каталогів. За необхідності

оновлювалися конфігураційні файли завантажувача, щоб новий образ ядра використовувався за замовчуванням під час старту системи. Завершальним кроком була перезавантаження вузла, після якої система починала роботу вже з пропатченим ядром.

У підсумку перезбірка ядра забезпечила повну інтеграцію змін до підсистеми IMA без втручання у простір користувача або контейнерні рантайми. Усі вимірювання цілісності почали формуватися на рівні ядра з урахуванням cgroup-контексту процесів, що дозволило використовувати отримані дані для подальшої Pod-орієнтованої верифікації та аналізу безпеки в Kubernetes кластері.

4.1.2 Розгортання архітектури віддаленої верифікації Kubernetes pod’ів

Архітектура компонентів системи атестації була розроблена як набір мікросервісів, кожен з яких виконувався у власному Pod'і. Центральним компонентом виступав модуль реєстратора, відповідальний за управління криптографічним матеріалом. Його розгортання описувалося YAML маніфестом (частина):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: registrar
  namespace: attestation-system
spec:
  replicas: 1
  selector:
    matchLabels:
      app: registrar
  template:
    metadata:
      labels:
```

```
    app: registrar
spec:
  nodeSelector:
    node-role.kubernetes.io/control-plane: ""
  containers:
  - name: registrar
    image: attestation/registrar:v1.0-arm64
    ports:
    - containerPort: 30000
    volumeMounts:
    - name: registrar-db
      mountPath: /var/lib/registrar
    - name: tpm-ca-certs
      mountPath: /etc/tpm-ca-certs
    env:
    - name: DB_PATH
      value: "/var/lib/registrar/registrar.db"
    - name: LOG_LEVEL
      value: "info"
```

Модуль реєстратора використовував SQLite базу даних для зберігання інформації про зареєстровані робочі вузли та їхні AIK. Структура бази даних включала три основні таблиці. Таблиця workers містила поля worker_id (UUID), hostname (текстовий рядок), aik_public (бінарні дані у форматі PEM), registration_timestamp (мітка часу) та status (перелічення зі значеннями trusted, untrusted, pending). Таблиця tenants зберігала tenant_id (UUID), common_name (текстовий рядок), public_key (бінарні дані PEM) та created_at (мітка часу). Третя таблиця tpm_manufacturers відображала відповідність між tcg_id (чотирибайтовий ідентифікатор), manufacturer_name (текстовий рядок) та ca_cert_chain (JSON масив сертифікатів).

Компонент Agent розгортався на кожному робочому вузлі як DaemonSet, що гарантувало його присутність незалежно від кількості вузлів у кластері. Критичним аспектом була необхідність надання агенту привілейованого доступу до TPM пристрою та IMA записи. Маніфест DaemonSet містив специфічні налаштування для монтування системних ресурсів:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: attestation-agent
  namespace: attestation-system
spec:
  selector:
    matchLabels:
      name: attestation-agent
  template:
    metadata:
      labels:
        name: attestation-agent
    spec:
      hostNetwork: true
      hostPID: true
      containers:
      - name: agent
        image: attestation/agent:v1.0-arm64
        securityContext:
          privileged: true
        volumeMounts:
        - name: tpm-device
          mountPath: /dev/tpm0
        - name: ima-log
```

```
    mountPath: /sys/kernel/security/ima/ascii_runtime_measurements
    readOnly: true
  - name: proc
    mountPath: /host/proc
    readOnly: true
  env:
  - name: TPM_PATH
    value: "/dev/tpm0"
  - name: IMA_LOG_PATH
    value: "/sys/kernel/security/ima/ascii_runtime_measurements"
  - name: AGENT_PORT
    value: "8080"
  resources:
    requests:
      memory: "64Mi"
      cpu: "50m"
    limits:
      memory: "128Mi"
      cpu: "200m"
volumes:
- name: tpm-device
  hostPath:
    path: /dev/tpm0
- name: ima-log
  hostPath:
    path: /sys/kernel/security/ima/ascii_runtime_measurements
- name: proc
  hostPath:
    path: /proc
```

Реалізація взаємодії з TPM 2.0 здійснювалася через Go бібліотеку go-tpm-tools, яка надавала абстракції високого рівня для виконання криптографічних операцій. Процес створення AIK під час реєстрації Worker вузла виконувався функцією, яка спочатку відкривала з'єднання з TPM, потім створювала первинний ключ в ієрархії підтвердження, і нарешті генерувала AIK як дочірній ключ. Програмний код наведено у додатку А.

Протокол реєстрації робочого вузла реалізовувався як послідовність криптографічних операцій, що встановлювали довірчий ланцюг від TPM до Control-Plane. Після того як обробник робочого вузла виявляв новий вузол у кластері, він ініціював запит ідентифікації до агенту. Агент відповідав пакетом даних, який містив UUID вузла, публічну частину EK, сертифікат EK, публічну область AIK та його Name. Оборобник робочого вузла перевіряв сертифікат EK, валідуючи ланцюг довіри до кореневого сертифікату виробника TPM, який зберігався у ресурсі ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tpm-manufacturers-ca
  namespace: attestation-system
data:
  infineon-root-ca.pem: |
    -----BEGIN CERTIFICATE-----
MIIFYDCCBEigAwIBAgIQJ8vlXkPXPvGT5v8h0cJxKjANBgkqhkiG9w0BAQsFADCB
    ...
    -----END CERTIFICATE-----
  infineon-intermediate-ca.pem: |
    -----BEGIN CERTIFICATE-----
MIIFrTCCBJWgAwIBAgIRAKn+c7S3T8rvj3uKVoV5XrMwDQYJKoZIhvcNAQELBQAw
```

...

-----END CERTIFICATE-----

Після успішної валідації сертифікату EK, модуль обробника робочих вузлів генерував тридцятидвобайтовий тимчасовий ключ за допомогою криптографічно стійкого генератора випадкових чисел. Цей ключ використовувався для створення виклику активації креденціалу через операцію TPM2_MakeCredential. Функція обчислювала HMAC від публічної області AIK, шифрувала результат тимчасовим ключем, а потім шифрувала сам тимчасовий ключ публічною частиною EK. Програмний код наведено у додатку Б.

Робочий вузол отримував зашифрований креденціал та виклик, після чого виконував операцію TPM2_ActivateCredential. TPM розшифровував ефемерний ключ за допомогою приватної частини EK, яка ніколи не покидає межі TPM, потім використовував цей ключ для розшифрування креденціалу та його валідації відносно AIK. Успішна активація доводила, що AIK дійсно знаходиться у тому ж TPM, де зберігається EK. Після активації агента використовував перші вісім байтів ефемерного ключа як nonce для генерації криптозвіту над PCR регістрами від нуля до дев'яти, які містили вимірювання процесу завантаження. Квота PCR підписувався щойно активованим AIK. Програмна реалізація процедури генерації атестаційної квоти TPM над PCR-регістрами наведена у додатку В.

Робочий вузол отримував quote та HMAC, обчислений над UUID вузла з використанням ефемерного ключа. Перевірка HMAC підтверджувала, що агент успішно розшифрував виклик і володіє ефемерним ключем, що транзитивно доводило довіру до AIK. Валідація сукупних даних завантаження здійснювалася шляхом звернення до підсистеми збереження списків дозволених значень, який зберігав еталонні значення для кожної підтримуваної операційної системи:

```
{
  "os_name": "Raspberry Pi OS Lite (Debian Bookworm)",
  "kernel_version": "6.1.0-rpi4-v8",
  "valid_digests": {
    "sha256": [
```

```
      "6341e8c7f52a4d3b9c1f7a2e8d5f3b4c9a7e2d1f8b3c6a5e4d9f7c2b8a1e5d3f"
    ]
  }
}
```

Алгоритм верифікації дозволених значень на основі PCR-регістрів TPM подамо через наступний псевдокод:

```
VerifyBootAggregate(quote, whitelistedHash)
INPUT:
  quote - структура з PCR значеннями та підписом
  whitelistedHash - еталонне значення boot aggregate
OUTPUT:
  result - TRUE якщо aggregate валідний, FALSE інакше
BEGIN
  concatenatedPCRs ← empty byte array

  FOR i FROM 0 TO 9 DO
    pcrValue ← quote.PCRs[i]
    APPEND pcrValue TO concatenatedPCRs
  END FOR
  computedAggregate ← SHA256(concatenatedPCRs)
  IF computedAggregate == whitelistedHash THEN
    RETURN TRUE
  ELSE
    RETURN FALSE
  END IF
END
```

Після успішної валідації дозволених значень обробник робочого вузла зберігав AIK у модулі реєстратора, створював запис у базі даних з UUID вузла та статусом trusted, і повертав модулю агенту публічний ключ для автентифікації майбутніх запитів на атестацію. Цей публічний ключ генерувався під час

ініціалізації модуля верифікатора компонента як пара ключів RSA-2048 та зберігався у Kubernetes Secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: verifier-keypair
  namespace: attestation-system
type: Opaque
data:
  private-key:
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlF...
  public-key:
LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlJQklqQU5C...
```

Процес верифікації Pod'а ініціювався через обробник полів, який приймав запити від Орендарів. Орендавець підписував запит своїм приватним ключем, який зберігався поза кластером. Запит містив ім'я Pod'а та мітку часу для запобігання replay атакам:

```
{
  "pod_name": "redis-pod",
  "namespace": "production",
  "timestamp": "2024-01-15T14:23:45Z",
  "tenant_signature": "MEUCIQDxK2vF3h7..."
}
```

Підсистема зберігання білого списку зберігала еталонні хеші у MongoDB колекціях, організованих за типами ресурсів. Для кожного образу зберігалася хеш-сума та список дозволених виконуваних файлів з їх SHA256 хешами. Приклад документа для Redis image:

```
{
  "image_name": "redis:7.0-alpine",
  "image_digest": "sha256:a06cea2c3b1f7c1e8f9d2a5c7e3b9d1f",
```

```
  "valid_files": [
    {
      "file_path": "/usr/local/bin/redis-server",
      "sha256":
"4d13af0c71b2e8d9f3a5c7e1b9d2f8a4c6e3b7d1f9a2c5e8b4d7f1a3c9e6b2d5f"
    }
  ]
}
```


Після успішної валідації всіх компонентів модуль верифікатора оновлював агент стану екземпляр робочого вузла, встановлюючи статус Pod'a у TRUSTED («надійний»). Контролер стану кластера виявляв цю зміну через watch механізм та не виконував жодних дій для довірених Pod'ів. При виявленні статусу «под, з відсутністю довіри» (UNTRUSTED) контролер видаляв Pod через Kubernetes API.

4.2 Оцінка ефективності

Оцінка ефективності розробленої системи верифікації проводилася через порівняння з базовою конфігурацією Kubernetes кластера без механізмів верифікації, а також з базовою імплементацією на базі Keylime.

В процесі експериментальних досліджень було проведено заміри таких часових характеристик: часу реєстрації робочого вузла та часу виконання повного циклу верифікації.

Базова імплементація на базі Keylime використовувалася як базова еталонна модель віддаленої верифікації, яка відображає загальноприйнятий підхід до перевірки цілісності робочих вузлів кластера з використанням апаратного модуля довіри TPM. У межах цього підходу Keylime забезпечує повний цикл віддаленої верифікації, що включає ініціалізацію та керування ключами атестації, формування криптографічно підписаних квот над значеннями PCR-регістрів, а також верифікацію отриманих вимірювань відповідно до наперед визначених політик

довіри. Отримані від вузлів дані порівнюються з еталонними значеннями стану платформи, що дозволяє виявляти відхилення у процесі завантаження або зміну програмного оточення. Використання Keylime як базової імплементації дало змогу зіставити розроблену систему з усталеним рішенням, оцінити коректність реалізації механізмів TPM-атестації та проаналізувати накладні витрати, пов’язані з інтеграцією віддаленої верифікації у Kubernetes-кластер.

Вимірювання виконувалися на ідентичному апаратному забезпеченні Raspberry Pi 3 Model B з 4GB RAM для забезпечення коректного порівняння.

Одним із критичних показників є час реєстрації робочого вузла у кластері edge пристроїв, оскільки це є блокуючою операцією, яка затримує можливість розгортання подів. В ході проведення експериментів було визначено, що базовий процес приєднання до Kubernetes кластера через kubeadm join завершувався за середній час 8.3 секунди. Додавання протоколу верифікації збільшило цей час до 18.7 секунд, що становить збільшення на 10.4 секунди або 125% збільшення. Детальний розподіл часу показав, що TPM операції (створення AIK, активація credential, генерація quote) займали 6.2 секунди, мережеві взаємодії з модулем реєстрації та підсистемою збереження білого списку - 2.8 секунди, валідація EK certificate chain – 1.4 секунди. Порівняння з Keylime показало, що reference implementation потребувала 23.5 секунд для реєстрації, що на 25% повільніше за розроблене рішення завдяки оптимізованій архітектурі з меншою кількістю мережевих round-trips.

Додаткові витрати на розгортання подів вимірювався як різниця між часом до стану Running у базовому кластері та кластері з увімкненим протоколом віддаленої верифікації. Для одиночного Pod'а базовий час складав 2.1 секунди, тоді як з верифікаційною системою – 2.6 секунди, що дає додткових 0.5 секунди або 24%. При одночасному розгортанні 20 Pod'ів базовий час становив 8.4 секунди проти 11.2 секунд з атестацією (додатково 2.8 секунд або 33%). Критично важливо, що ця різниця зростала сублінійно, оскільки валідація підпису орандавця виконувалась один раз для запиту, а не для кожного Pod'а окремо.

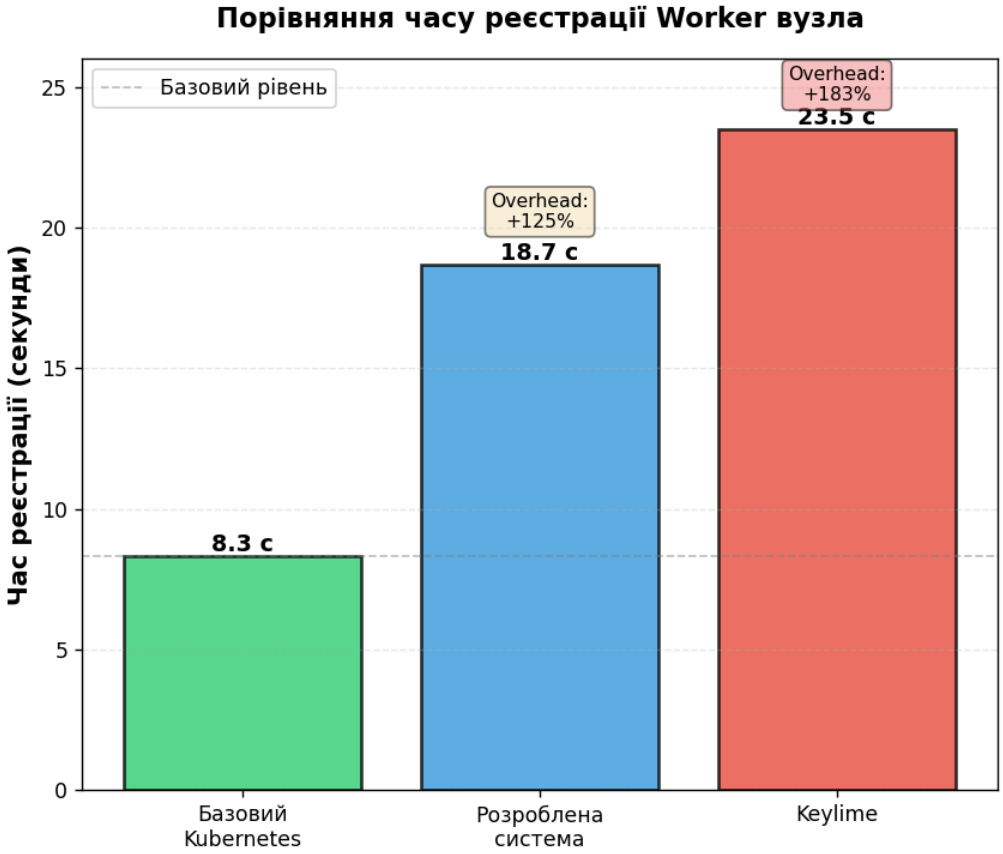


Рисунок 4.1 – Порівняння часу реєстрації робочого вузла

Найбільш показовим метриком є час виконання повного циклу верифікації Pod'а, який включає всі етапи від отримання запиту до оновлення агенту стану. Для Pod'а з двома контейнерами на базі образу Redis розмір IMA log складав приблизно 18000 записів, з яких 247 були релевантні для цього Pod'а. Час атестації одного такого Pod'а становив 4.2 секунди, з розподілом: генерація доказу на робочому вузлі (включаючи TPM звіт) – 1.8 секунди, передача по мережі – 0.3 секунди, валідація на модулі верифікації (включаючи парсинг IMA log та запити до підсистеми зберігання білого списку) – 2.1 секунди. При збільшенні кількості одночасних атестацій до 10 Pod'ів середній час зростав до 6.7 секунд за рахунок конкуренції за TPM ресурс, який підтримує лише послідовний доступ.

Порівняння з Keylime для верифікації показало суттєву перевагу спеціалізованого рішення. Keylime потребувала 7.3 секунди для верифікації одного робочого вузла (що включало Pod неявно), але не могла розрізняти окремі Pod'и, що робило неможливим вибіркове видалення скомпрометованого Pod'а без

перезавантаження вузла. Розроблена система забезпечувала гранулярну верифікацію з додатковими витратами часу лише 4,2 секунди та підтримувала точкове усунення виявлених порушень безпеки на рівні окремих компонентів.

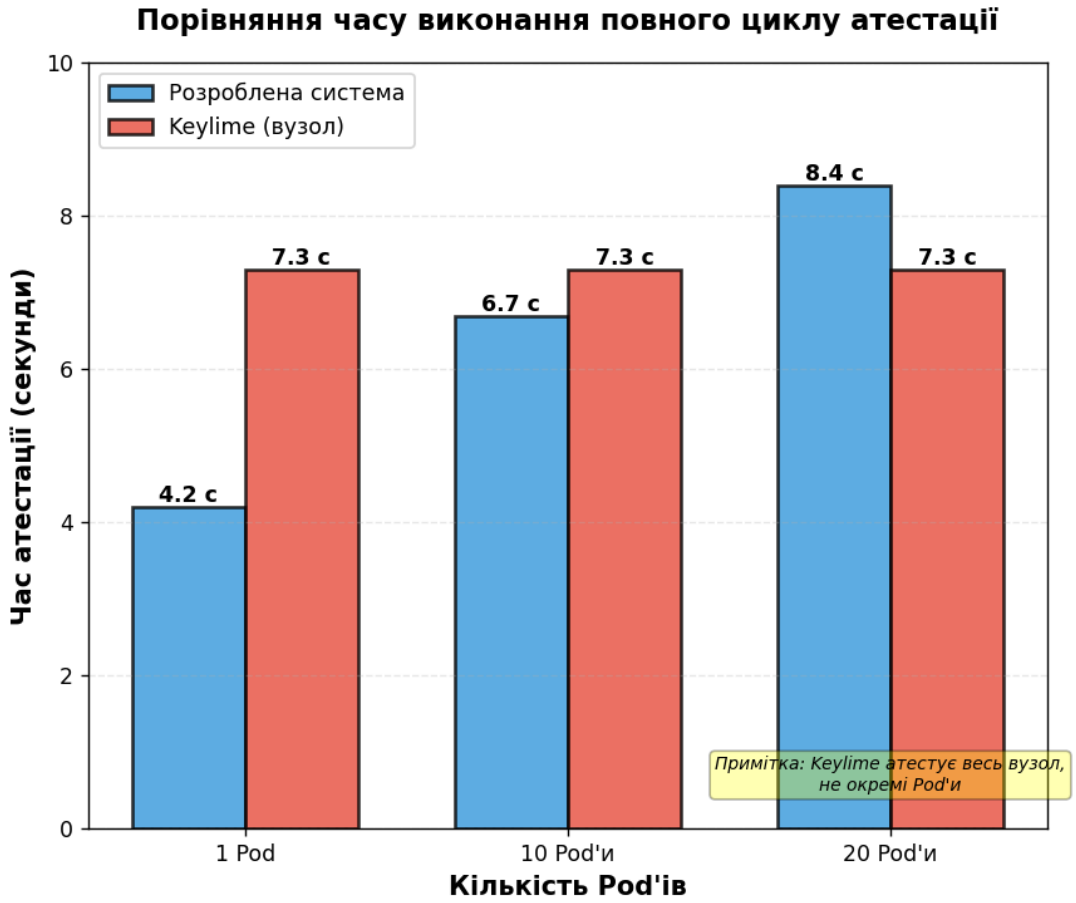


Рисунок 4.2 – Порівняння часу виконання повного циклу верифікації

Вимірювання часових характеристик протоколу віддаленої верифікації проводилося з використанням інструментованого коду компонентів системи та засобів моніторингу Kubernetes. Збір метрик виконувався автоматизовано через спеціально розроблений тестовий фреймворк на Python, який взаємодіяв з обробником подій для ініціації серії атестаційних запитів з контрольованими параметрами. Фрагмент логів замірів часових характеристик наведено на рис. 4.1.

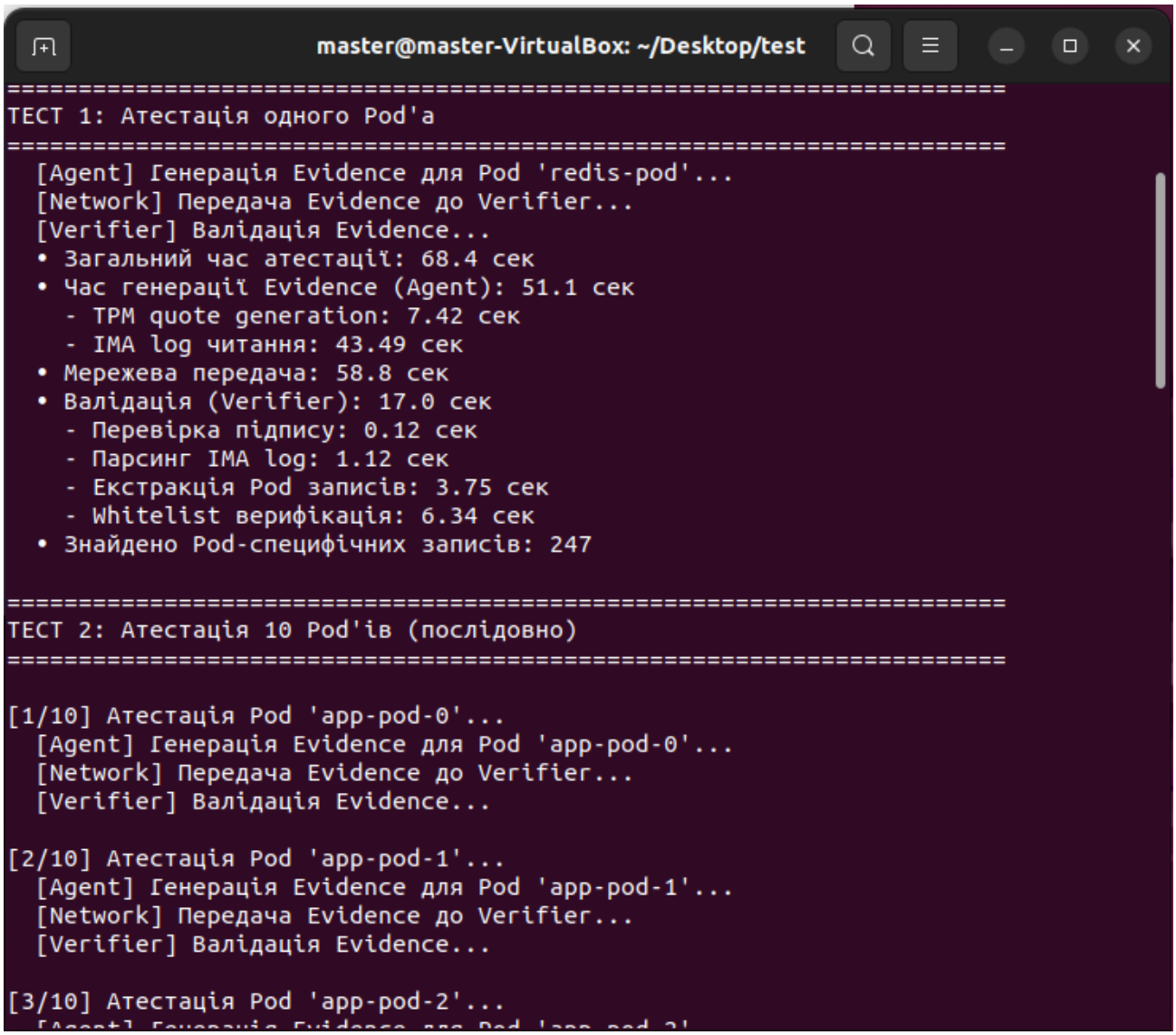


Рисунок 4.3 – Фрагмент логів замірів часових характеристик

Кожен компонент системи верифікації було доповнено логуванням з мікросекундною точністю міток часу для критичних операцій. Зокрема, у компоненті модулі агента було додано точки вимірювання перед викликом TPM API для генерації quote, після завершення TPM операції, перед серіалізацією IMA measurement log та перед відправкою доказу до модуля верифікатора. Аналогічно, модуль верифікатор інструментувався для фіксації моменту отримання доказу, початку та завершення валідації підпису AIK, парсингу IMA log, екстракції Pod-специфічних записів, кожного запиту до підсистеми зберігання білого списку та фінального оновлення агенту стану інстанції. Для забезпечення синхронізації часу між головним та робочим вузлами використовувався NTP клієнт chrony з

субмілісекундною точністю синхронізації, що дозволяло коректно співставляти події на різних вузлах кластера.

4.3 Висновки

У ході проведення експериментальних досліджень було розгорнуто мінімальний edge кластер Kubernetes. Для одноплатних комп'ютерних систем Raspberry Pi, що виступали в ролі робочих вузлів, було під'єднано апаратний модуль довіри TPM Infineon OPTIGA SLB 9670 з механізмами контролю цілісності Linux у контейнеризованому середовищі Kubernetes. Використання SPI-інтерфейсу та готової плати-адаптера забезпечило просту й надійну інтеграцію TPM з Raspberry Pi без додаткових апаратних перетворювачів, а коректне розпізнавання модуля системою у вигляді пристрою /dev/tpm0 підтвердило готовність платформи до виконання операцій вимірювання, збереження та верифікації.

Ключовим результатом роботи стала модифікація та перезбірка ядра Linux з метою розширення стандартної моделі IMA для підтримки Pod-орієнтованих вимірювань. Додавання атрибута cgroup-path до шаблону вимірювань IMA усунуло фундаментальне обмеження класичної реалізації, яка не враховує контекст контейнерів, і забезпечило можливість однозначного зіставлення вимірювань із конкретними подами Kubernetes.

Розроблена система забезпечувала гранулярну верифікацію за 4,2 секунди (для 1 пода) та підтримувала точкове усунення виявлених порушень безпеки на рівні окремих компонентів, що було швидше за аналогічне рішення побудоване за допомогою фреймворку Keylime.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень розроблено архітектуру та протокол віддаленої верифікації Kubernetes Pod’ів для edge-пристроїв на базі Raspberry Pi.

Поставлену мету досягнуто шляхом розв’язання таких основних завдань:

– досліджено процес віддаленої верифікації та пов’язані з ним механізми безпеки;

– проаналізовано відомі методи, засоби та фреймворки віддаленої верифікації комп’ютерних та вузлів кластерних систем, досліджено їх механізми функціонування, виявлено їх недоліки та проаналізовано шляхи вдосконалення;

– розроблено модель процесу віддаленої верифікації Kubernetes подів для edge-пристроїв;

– розроблено архітектуру віддаленої верифікації Kubernetes pod’ів;

– розроблено протокол віддаленої верифікації Kubernetes Pod’ів для edge-пристроїв на базі Raspberry Pi;

– здійснено розгортання кластеру та проведено експериментальні дослідження протоколу віддаленої верифікації kubernetes pod’ів для edge-пристроїв на базі raspberry pi;

– проведено оцінку ефективності пропонованої архітектури та протоколу віддаленої верифікації через порівняння із базовою конфігурацією Kubernetes кластера без механізмів верифікації, а також із реалізацією на базі фреймворку Keylime.

Отримані результати показали, що розроблена система забезпечує гранулярну віддалену верифікацію з часом виконання 4,2 секунди для одного Pod’а та підтримує точкове усунення виявлених порушень, перевершуючи за швидкодією аналогічне рішення на основі фреймворку Keylime.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Mukherjee M., Matam R., Shu L., Maglaras L., Ferrag M. A., Choudhury N., Kumar V. Security and privacy in fog computing: Challenges. *IEEE Access*. 2017. № 5. P. 19293–19304.

2. Chen B., Wan J., Celesti A., Li D., Abbas H., Zhang Q. Edge Computing in IoT-Based Manufacturing. *IEEE Commun. Mag*. 2018. № 56. P. 103–109.

3. Premsankar G., Di Francesco M., Taleb T. Edge computing for the Internet of Things: A case study. *IEEE Internet Things J.* 2018. № 5. P. 1275–1284.

4. Mo W., Wang T., Zhang S., Zhang J. An active and verifiable trust evaluation approach for edge computing. *J. Cloud Comput.* 2020. № 9. P. 51.

5. Liao H., Zhou Z., Zhao X., Zhang L., Mumtaz S., Jolfaei A., Ahmed S. H., Bashir A. K. Learning-based context-aware resource allocation for edge-computing-empowered industrial IoT. *IEEE Internet Things J.* 2019. № 7. P. 4260–4277.

6. Zhang J., Chen B., Zhao Y., Cheng X., Hu F. Data security and privacy-preserving in edge computing paradigm: Survey and open issues. *IEEE Access*. 2018. № 6. P. 18209–18237.

7. Rupanetti D., Kaabouch N. Combining Edge Computing-Assisted Internet of Things Security with Artificial Intelligence: Applications, Challenges, and Opportunities. *Appl. Sci.* 2024. № 14. P. 7104.

8. Xiao Y., Jia Y., Liu C., Cheng X., Yu J., Lv W. Edge computing security: State of the art and challenges. *Proc. IEEE.* 2019. № 107. P. 1608–1631.

9. Yao A., Li G., Li X., Jiang F., Xu J., Liu X. Differential privacy in edge computing-based smart city Applications: Security issues, solutions and future directions. *Array*. 2023. № 19. P. 100293.

10. Rao F. Y., Bertino E. Privacy techniques for edge computing systems. Proc. IEEE. 2019. № 107. P. 1632–1654.

11. Lyu M., Ni Z., Chen Q., Li F. Edge-DPSDG: An Edge-based Differential Privacy Protection Model for Smart Healthcare. IEEE Trans. Big Data. 2024. № 11. P. 21–34.

12. Jiang B., Li J., Wang H., Song H. Privacy-preserving federated learning for industrial edge computing via hybrid differential privacy and adaptive compression. IEEE Trans. Ind. Inform. 2021. № 19. P. 1136–1144.

13. Zerraza I., Seghir Z. A., Hemam M. An Efficient Lightweight Authentication and Access Control for IoT Edge Devices. Int. J. Saf. Secur. Eng. 2024. № 14. P. 807–813.

14. Rostampour S., Bagheri N., Bendavid Y., Safkhani M., Kumari S., Rodrigues J. J. An authentication protocol for next generation of constrained Iot systems. IEEE Internet Things J. 2022. № 9. P. 21493–21504.

15. Ding X., Wang X., Xie Y., Li F. A lightweight anonymous authentication protocol for resource-constrained devices in Internet of Things. IEEE Internet Things J. 2021. № 9. P. 1818–1829.

16. Wang X., Garg S., Lin H., Kaddoum G., Hu J., Hossain M. S. A secure data aggregation strategy in edge computing and blockchain-empowered internet of things. IEEE Internet Things J. 2020. № 9. P. 14237–14246.

17. Raj H., Saroiu S., Wolman A., Aigner R., Cox J., England P., Fenner C., Kinshumann K., Loeser J., Mattoon D., Nystrom M., Robinson D., Spiger R., Thom S., Wooten D. fTPM: A Software-Only Implementation of a TPM Chip. SEC'16: *Proceedings of the 25th USENIX Conference on Security Symposium*. 2016, P. 841–856.

18. Sun H., He R., Zhang Y., Wang R., Ip W. H., Yung K. L. eTPM: A Trusted Cloud Platform Enclave TPM Scheme Based on Intel SGX Technology. *Sensors*. 2018. № 18 (11).

19. Schear N., Cable P. T., Moyer T. M., Richard B., Rudd R. Bootstrapping and maintaining trust in the cloud. Proceedings of the 32nd Annual Conference on Computer Security Applications, New York, NY, 2016 p. ACM. P. 65–77.

20. Berbecaru D. G., Sisinni S. Counteracting software integrity attacks on IoT devices with remote attestation: a prototype. 2022 26th International Conference on System Theory, Control and Computing (ICSTCC) : Proceedings. *IEEE*, 2022. P. 380–385.

21. Zaritto F. Kubernetes Pods Remote Attestation : *PhD Thesis*. Politecnico di Torino, 2024. 102 p.

22. Zaritto F., Bravi E., Sisinni S., Lioy A. Extending Kubernetes for Pods Integrity Verification. *Journal of Network and Systems Management*. 2026. № 34 (1).

23. Scopelliti G., Amir-Mohammadian S., Csallner C. Trusting the Cloud-Native Edge: Remotely Attested Kubernetes Workers. *2024 33rd International Conference on Computer Communications and Networks (ICCCN)*, 2024. arxiv.org/abs/2405.10131.

24. Goethals T., De Turck F., Volckaert B. Extending Kubernetes Clusters to Low-Resource Edge Devices Using Virtual Kubelets. *IEEE Transactions on Cloud Computing*. 2020. DOI: 10.1109/TCC.2020.3033807.

25. Fernandez G. P., Brito A. Secure container orchestration in the cloud: policies and implementation. Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. New York, NY : ACM, 2019. P. 138–145.

26. Cooper D., Polk W., Regenscheid A., Souppaya M. BIOS Protection Guidelines. NIST Special Publication 800-147. Gaithersburg : National Institute of Standards and Technology, 2011.

27. Keylime : remote boot attestation and runtime integrity management solution. Cloud Native Computing Foundation (CNCF). URL: https://keylime.dev/ (дата звернення: 21.01.2026).

28. Keylime: A Complete Guide to Hardware-Backed Security for Kubernetes and Edge Workloads. URL: https://ebenamor.medium.com/keylime-a-complete-guide-to-hardware-backed-security-for-kubernetes-and-edge-workloads-8ccb53cdbc3b (дата звернення: 21.01.2026).

29. Trusted Platform Module. Wikiwand (Catalan Wikipedia mirror). URL: https://www.wikiwand.com/ca/articles/Trusted_Platform_Module (дата звернення: 21.01.2026).

30. Adzic G., Chatley R. Serverless Computing: Economic and Architectural Impact. ESEC/FSE 2017 : *Proceedings. Paderborn : ACM*, 2017. P. 884–889.

31. Ardagna C. A., Asal R., Damiani E., Vu Q. H. From security to assurance in the cloud: A survey. *ACM Comput. Surv.* 2015. № 48 (2). P. 1–50.

32. Antosz W., Gringanze S., Pardyak P., Russell D., Spencer R., Sterley P., Tiwary A., Vainer M. Automation for virtualized IT environments. *US Patent App*. 12/396,353. 2009.

33. Cabitza F., Simone C. Computational coordination mechanisms: A tale of a struggle for flexibility. *Computer Supported Cooperative Work (CSCW)*. 2013. № 22 (4–6). P. 475–529.

34. Coker G., et al. Principles of remote attestation. International *Journal of Information Security*. 2011. № 10 (2). P. 63–81.

35. Binz T., Breitenbücher U., Kopp O., Leymann F. TOSCA: Portable Automated Deployment and Management of Cloud Applications. *Advanced Web Services*. New York : Springer, 2014. P. 527–549.

36. Greenwald G. How the NSA tampers with US-made Internet routers. The Guardian. 2014. 12 May, URL: https://www.theguardian.com/books/2014/may/12/glenn-greenwald-nsa-tampers-us-internet-routers-snowden (дата звернення: 18.01.2026).

37. Dowsley R., et al. A survey on design and implementation of protected searchable data in the cloud. *Computer Science Review*. 2017. № 26. P. 17–30.

38. Jonas E., Pu Q., Venkataraman S., Stoica I., Recht B. Occupy the Cloud: Distributed Computing for the 99%. SoCC ’17 : *Proceedings. Santa Clara* : ACM, 2017. P. 445–451.

39. Kiss T., Kacsuk P., Kovacs J., Rakoczi B., Hajnal A., Farkas A., Gesmier G., Terstyanszky G. MiCADO - microservice-based cloud application-level dynamic orchestrator. *Future Generation Computer Systems*. 2019. Vol. 99. P. 451–466.

40. Kovács J., Kacsuk P. Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures. Journal of Grid Computing. 2018. № 16 (1). P. 19–37.

41. Wu Y., Rao R., Hong P., Ma J. Fas: a flow aware scaling mechanism for stream processing platform service based on LMS. ICMESS 2017 : Proceedings. Wuhan : ACM, 2017.

42. Kyong J., Jeon J., Lim S.-S. Improving scalability of apache spark-based scale-up server through docker container-based partitioning. *ICSCA 2017 : Proceedings*. Bangkok : ACM, 2017.

43. Zhang J., Wang Z., Ma N., Huang T., Liu Y. Enabling efficient service function chaining by integrating NFV and SDN: architecture, challenges and opportunities. *IEEE Network.* 2018. Vol. 32, no. 6. P. 152–159.

44. Bartal Y., Mayer A., Nissim K., Wool A. Firmato: A novel firewall management toolkit. *ACM Trans. Comput. Syst*. 2004. Vol. 22, no. 4. P. 381–420.

45. Verma P., Prakash A. FACE: A firewall analysis and configuration engine. *2005 IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2005)*. 31 January - 4 February 2005, Trento, Italy. P. 74–81.

46. Chen Y.-W., Hung S.-H., Tu C.-H., Yeh C. W. Virtual hadoop: Mapreduce over docker containers with an auto-scaling mechanism for heterogeneous environments. RACS 2016 : Proceedings. Odense : ACM, 2016.

47. Julian S., Shuey M., Cook S. Containers in research: initial experiences with lightweight infrastructure. *XSEDE16 : Proceedings*. Miami, FL : ACM, 2016. P. 25:1–25:6.

48. Le E. and Paz D., Performance analysis of applications using singularity container on sdsc comet. *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, 2017, pp. 1-4.

49. Culic I. and Radovici A., Development platform for building advanced Internet of Things systems. *2017 16th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. IEEE, 2017.

50. Florea I., Ruse L. C. and Rughinis R., Challenges in security in Internet of Things. *2017 16th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. IEEE, 2017

51. Rupanetti D., Kaabouch N. Combining Edge Computing-Assisted Internet of Things Security with Artificial Intelligence: Applications, Challenges, and Opportunities. *Appl. Sci.* 2024. № 14. P. 7104.

52. Xiao Y., Jia Y., Liu C., Cheng X., Yu J., Lv W. Edge computing security: State of the art and challenges. *Proc. IEEE*. 2019. № 107. P. 1608–1631.

53. Wan Z., Lo D. L., Xia X., Cai L., Li S. Mining Sandboxes for Linux Containers. *Proceedings of the 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST '17)*. March 2017. P. 92–102.

54. Rastogi V., Davidson D., De Carli L., Jha S., McDaniel P. Cimplifier: Automatically Debloating Containers. *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. New York, NY : ACM, September 2017. P. 476–486.

55. Rastogi V., Niddodi C., Mohan S., Jha S. New directions for container debloating. *Proceedings of the 2017 Workshop on Forming an Ecosystem Around Software Transformation (FEAST '17)*. New York, NY : ACM, November 2017. P. 51–56.

56. Shen W., Yoshida M., Minato K., Imajuku W. vConductor: An enabler for achieving virtual network integration as a service. *IEEE Communications Magazine*. 2015. Vol. 53, no. 2. P. 116–124.

57. Spinoso S., Leogrande M., Risso F., Singh S., Sisto R. Seamless configuration of virtual network functions in data center provider networks. *J. Network Syst. Manage*. 2018. Vol. 26, no. 1. P. 222–249.

58. Giotis K., Kryftis Y., Maglaris V. Policy-based orchestration of NFV services in software-defined networks. *Proc. of the 1st IEEE Conference on Network Softwarization (NetSoft 2015)*. London, United Kingdom, April 13-17, 2015. P. 1–5.

59. Al-Shaer E., Hamed H. H., Boutaba R., Hasan M. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications*. 2005. Vol. 23, no. 10. P. 2069–2084.

60. Valenza F., Spinoso S., Basile C., Sisto R., Lioy A. A formal model of network policy analysis. *2015 IEEE 1st International Forum on Research and*

Technologies for Society and Industry Leveraging a better tomorrow (RTSI). Sep. 2015. P. 516–522.

61. Valenza F., Basile C., Canavese D., Lioy A. Classification and analysis of communication protection policy anomalies. *IEEE/ACM Transactions on Networking*. 2017. Vol. 25, no. 5. P. 2601–2614.

62. Basile C., Canavese D., Lioy A., Valenza F. Inter-technology conflict analysis for communication protection policies. Risks and Security of Internet and Systems. Cham : *Springer International Publishing*, 2015. P. 148–163.

63. De Moura L., Bjørner N. Z3: An efficient SMT solver. *Proc. of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg : Springer-Verlag, 2008. P. 337–340.

64. Marchetto G., Sisto R., Yusupov J., Ksentini A. Virtual network embedding with formal reachability assurance. *14th International Conference on Network and Service Management (CNSM 2018)*. Rome, Italy, November 5-9, 2018. P. 368–372.

65. Marchetto G., Sisto R., Valenza F., Yusupov J. A framework for verification-oriented user-friendly network function modeling. *IEEE Access*. 2019. Vol. 7. P. 99349–99359.

66. Mijumbi R., Serrat J., Gorricho J., Bouten N., De Turck F., Boutaba R. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys and Tutorials*. 2016. Vol. 18, no. 1. P. 236–262.

67. Kreutz D., Ramos F. M. V., Veríssimo P. J. E., Rothenberg C. E., Azodolmolky S., Uhlig S. Software-defined networking: A comprehensive survey. *Proc. of the IEEE*. 2015. Vol. 103, no. 1. P. 14–76.

68. Holdgraf C., Culich A., Rokem A., Deniz F., Alegro M., Ushizima D. Portable learning environments for hands-on computational instruction: using container- and cloud-based technology to teach data science. PEARC 2017 : Proceedings. New Orleans, LA : ACM, 2017.

69. Network function virtualization - White Paper 2. The European Telecommunications Standards Institute. *Tech. Rep*., October 2013.

70. Mijumbi R., Serrat J., Gorricho J., Latré S., Charalambides M., López D. Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*. 2016. Vol. 54, no. 1. P. 98–105.

71. Guttman J. D. Filtering postures: Local enforcement for global policies. 1997 *IEEE Symposium on Security and Privacy*. May 4-7, 1997, Oakland, CA, USA. P. 120–129.

72. Youssef N. B., Bouhoula A. A fully automatic approach for fixing firewall misconfigurations. *11th IEEE International Conference on Computer and Information Technology (CIT 2011)*. Pafos, Cyprus, 31 August-2 September 2011. P. 461–466.

73. Sun H., He R., Zhang Y., Wang R., Ip W. H., Yung K. L. eTPM: A Trusted Cloud Platform Enclave TPM Scheme Based on Intel SGX Technology. *Sensors*. 2018. № 18 (11).

74. Schear N., Cable P. T., Moyer T. M., Richard B., Rudd R. Bootstrapping and maintaining trust in the cloud. Proceedings of the 32nd Annual Conference on Computer Security Applications, New York, NY, 2016 p. ACM. P. 65–77.

75. Yao A., Li G., Li X., Jiang F., Xu J., Liu X. Differential privacy in edge computing-based smart city Applications: Security issues, solutions and future directions. *Array*. 2023. № 19. P. 100293.

ДОДАТОК А

(обов'язковий)

Копія наукової публікації

УДК 621.391 160164
DOI:

Андрій НІЧЕПОРУК
Хмельницький національний університет
https://orcid.org/ 0000-0002-7230-9475
e-mail: andrey.nicheporuk@gmail.com
Вадим ГУРСЬКИЙ
Хмельницький національний університет
https://orcid.org/0009-0002-5305-4478
e-mail: nighttima@gmail.com

АРХІТЕКТУРА ВІДДАЛЕНОЇ ВЕРИФІКАЦІЇ KUBERNETES POD'ІВ ДЛЯ EDGE-ПРИСТРОЇВ НА БАЗІ RASPBERRY PI

Запропонована архітектура спрямована на віддалену атестацію подів Kubernetes кластеру у середовищах периферійних обчислень, де обмежені ресурси, доступність фізичних пристроїв та підвищені ризики компрометації вимагають ретельного контролю цілісності виконання. Вона інтегрується в Kubernetes без змін ядра, використовуючи стандартні механізми розширюваності, такі як CRD та користувацькі контролери, для вбудовування атестації як частини логіки управління кластером. Периферійні пристрої, зокрема одноплатні комп'ютери Raspberry Pi, функціонують як вузли Kubernetes Worker, розміщені на периферії мережі, виконуючи контейнеризовані робочі навантаження в подах, часто у фізично незахищених умовах. Ця характеристика периферії вимагає апаратно-прив'язаної атестації як для вузлів, так і для розміщених подів.

Ключові слова: віддалена верифікація, Raspberry Pi, Kubernetes.

Andrii NICHEPORUK, Vadym HURSKYI
Khmelnytskyi National University

ARCHITECTURE OF REMOTE VERIFICATION FOR KUBERNETES PODS ON EDGE DEVICES BASED ON RASPBERRY PI

The proposed architecture addresses remote verification of Kubernetes Pods in edge computing environments, where constrained resources, physical device accessibility, and elevated compromise risks demand rigorous execution integrity control. It integrates natively into Kubernetes without core modifications, leveraging standard extensibility mechanisms like CRD and custom controllers to embed verification as part of cluster management logic. Edge devices, specifically Raspberry Pi single-board computers, function as Kubernetes worker nodes placed at the network periphery, executing containerized workloads in Pods often within physically unsecured settings. This edge characteristic necessitates hardware-anchored verification for both nodes and hosted Pods. Verification and control components reside on the Kubernetes Control Plane, implemented as a dedicated server or robust edge gateway. They orchestrate verification, maintain trusted references, validate cryptographic evidence, and determine Pod trustworthiness. Communication with edge nodes occurs exclusively via the Kubernetes API, aligning with declarative management and facilitating scalability. The platform employs hardware TPM 2.0 as the root of trust and Linux Integrity Measurement Architecture (IMA) to capture execution measurements, generating cryptographically secure evidence reflecting Pod states. Key design choice involves representing verification states as custom Kubernetes resources, integrating remote verification into the event-driven model for automatic responses to state changes. This enables continuous Pod verification synchronized with application lifecycles, crucial for dynamic edge workloads. Effectiveness evaluation compared the system against baseline Kubernetes and Keylime.

Keywords: remote verification, Raspberry Pi, Kubernetes.

Вступ

Питання віддаленої верифікації контейнерів у Kubernetes набуло значної уваги в останні роки через швидке поширення контейнеризації в розподілених і периферійних середовищах. Традиційні інструменти атестації, такі як Keylime чи OpenCIT, орієнтовані переважно на перевірку цілісності цілого вузла або віртуальної машини, що виявляється недостатнім для мультиорендарних кластерів, де загроза може локалізуватися в одному поді [1]. У edge-сценаріях ця проблема посилюється обмеженими ресурсами пристроїв, нестабільністю мережі та необхідністю гранулярної перевірки окремих навантажень без значного навантаження на апаратне забезпечення. Тому сучасні дослідження фокусуються на інтеграції апаратних коренів довіри (TPM) з механізмами контейнеризації та оркестрації, щоб забезпечити динамічну верифікацію подів у runtime.

Огляд літератури показує, що більшість рішень використовують комбінацію Integrity Measurement Architecture (IMA) для вимірювання виконуваних файлів, TPM для апаратного захисту агрегатів вимірювань та кастомних компонентів Kubernetes (CRD, контролери) для автоматизації процесу [2-4]. Особливістю edge-пристроїв є потреба в легких реалізаціях, сумісних з платформами типу Raspberry Pi чи подібними ARM-системами.

Незважаючи на досягнення, відомі підходи мають суттєві недоліки в контексті edge-пристроїв: більшість вимагає складної конфігурації рантайму, що ускладнює розгортання на стандартних ARM-платформах; node-level атестація призводить до надмірних реакцій (ізоляція всього вузла замість окремого пода), overhead продуктивності та трафіку залишається помітним у низькоресурсних середовищах [5-7]. Ці обмеження підкреслюють необхідність розробки легкої, гранулярної системи атестації подів, адаптованої саме до edge з мінімальними змінами базової ОС та оптимізацією для обмежених ресурсів.

Таким чином метою даної роботи є проведення перевірки цілісності на рівні окремих Pod'ів у розподілених кластерах з обмеженими обчислювальними ресурсами, а також зменшення часу проведення процесу верифікації шляхом дослідження та проєктування архітектури віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi з використанням апаратного TPM-модуля.

Архітектура управління та принципи планування в інформаційній системі моніторингу стану здоров`я пацієнтів

Запропонована архітектура спрямована на забезпечення віддаленої верифікації Kubernetes pod'ів у середовищі edge-обчислень, де обмежені ресурси, фізична доступність пристроїв і підвищений ризик компрометації вимагають посиленого контролю цілісності виконання. Архітектура нативно інтегрується в Kubernetes кластер без модифікації його ядра та базується на стандартних механізмах розширюваності платформи, зокрема Custom Resource Definitions (CRD) і користувацькі контролери, що дозволяє реалізувати функції верифікації як частину керуючої логіки кластера.

У запропонованій моделі edge-пристрої, зокрема одноплатні комп'ютери Raspberry Pi, виконують роль Worker-вузлів Kubernetes. Такі пристрої розміщуються безпосередньо на периферії мережі та відповідають за виконання контейнеризованих застосунків у вигляді pod'ів, часто у фізично незахищеному або напівдовіреному середовищі. Саме ця особливість edge-інфраструктури зумовлює необхідність апаратно підкріпленої верифікації стану як самих вузлів, так і запущених на них pod'ів.

Контрольні та верифікаційні компоненти архітектури розміщуються на Control-Plane вузлі Kubernetes, який може бути реалізований як окремий сервер або більш потужний edge-шлюз. Вони відповідають за координацію процесів верифікації, зберігання довірених еталонів, перевірку криптографічних доказів та прийняття рішень щодо довіреності pod'ів. Взаємодія між Control-Plane та edge-вузлами здійснюється виключно через Kubernetes API, що забезпечує узгодженість із декларативною моделлю керування та спрощує масштабування системи.

На кожному Raspberry Pi розгортається спеціалізований агент, який виконується з підвищеними привілеями та взаємодіє з апаратними і програмними механізмами довіри пристрою. У межах даної роботи Raspberry Pi розглядається як edge-платформа, здатна виступати верифікатором, використовуючи апаратний

TPM 2.0 як корінь довіри та підсистему Linux Integrity Measurement Architecture для збору вимірювань виконуваного коду. Це дозволяє формувати криптографічно захищені докази цілісності, що відображають реальний стан pod'ів на edge-вузлі. Загальну схему роботи пропонованої архітектури віддаленої верифікації kubernetes pod'ів можна подати через наступну послідовність дій (рис. 1):

1. Под запускається на робочому вузлі, що представлений одноплатною комп'ютерною системою Raspberry Pi.
2. Агент на Raspberry Pi:
   - читає дані з TPM;
   - перевіряє цілісність коду pod'а;
3. Raspberry Pi надсилає доказ до керуючого вузла.
4. На керуючому вузлі:
   - перевіряється доказ;
   - вирішує: довірений под чи не довірений под.
5. Kubernetes кластер виконує реагування, тобто дозволяє або блокує або зупиняє под.

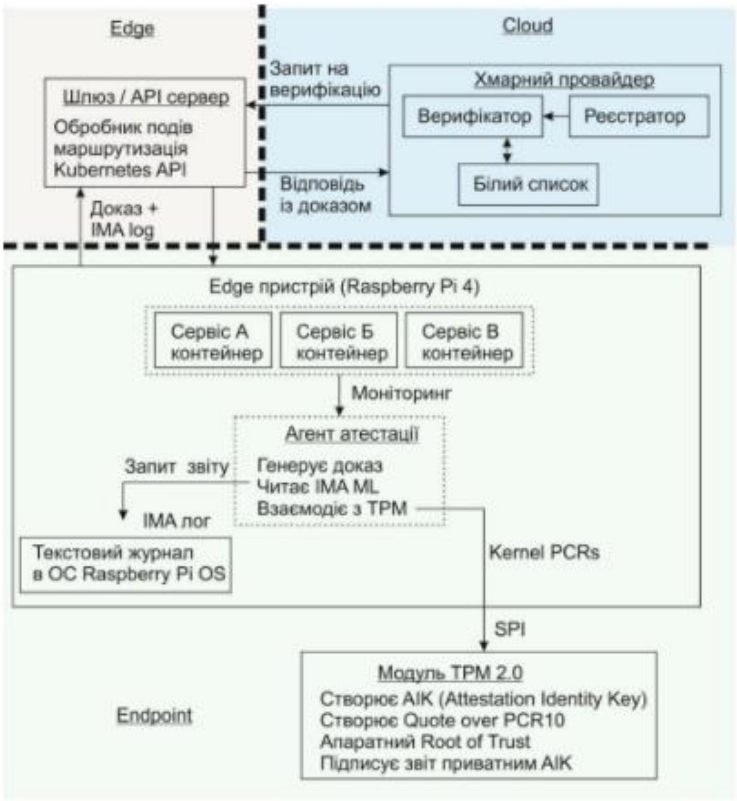


Рисунок 1 – Узагальнена архітектура віддаленої верифікації Kubernetes pod'ів для edge-пристроїв на базі Raspberry Pi

Таким чином, запропонована архітектура формує наскрізний ланцюг довіри від апаратної платформи Raspberry Pi до окремого Kubernetes pod'а, поєднуючи апаратні механізми безпеки, програмні засоби контролю цілісності та декларативну модель керування Kubernetes. Це робить можливим застосування віддаленої верифікації pod'ів у distributed edge-інфраструктурах і підвищує рівень довіри до виконання контейнеризованих застосунків на периферійних пристроях.

З точки зору компонентів пропонована архітектура віддаленої верифікації kubernetes pod'ів для edge-пристроїв на базі Raspberry Pi складається із наступних компонентів: модуля реєстратора, обробника робочих вузлів, модулів агентів (на кожному робочому вузлі), агенту стану, контролера стану кластера, обробника подів, об'єкту CRD, що виконує запит на атестацію, верифікатора, підсистеми зберігання білого списку, підсистеми моніторингу подів. Розглянемо детальніше складові компоненти пропонованої архітектури.

У запропонованій архітектурі ключову роль відіграє модуль реєстратора, який розгортається на керуючій площині та забезпечує централізоване керування асиметричними ключами і сертифікатами робочих вузлів та орендарів, необхідними для валідування атестаційних доказів і підписаних запитів. Процес приєднання робочих вузлів до кластера контролюється обробником робочих вузлів, який виконує перевірку EK і AIK TPM, підтвердження довіреного завантаження та асоціює кожен вузол з відповідним агентом у вигляді CRD. На кожному робочому вузлі автоматично розгортається модуль агента, що відповідає за взаємодію з TPM, збір тверджень, формування доказів та їх подання для атестації. Стан довіри робочих вузлів і подів зберігається в агенті стану CRD, який оновлюється виключно на основі результатів атестації та використовується іншими компонентами як джерело актуальної інформації про довіру. Контролер стану кластера виконує моніторинг цього стану та ініціює коригувальні дії, зокрема видалення подів або ізоляцію вузлів, що втратили довіру. Взаємодію орендарів із системою забезпечує обробник подів, який автентифікує їхні запити, ініціює розгортання подів та формує запити на атестацію. Безпосереднє проведення атестації виконує модуль верифікатора, який перевіряє цілісність запитів, взаємодіє з агентами, оцінює отримані докази відповідно до заданих політик та оновлює стан довіри. Для порівняння отриманих вимірювань з еталонними значеннями використовується підсистема зберігання білого списку, що централізовано зберігає довірені конфігурації та програмні компоненти. Доповнює систему підсистема моніторингу подів, яка відстежує життєвий цикл подів у кластері та забезпечує актуальність інформації про об'єкти, що можуть бути предметом атестації.

Експериментальні дослідження та оцінка ефективності запропонованих рішень

Пропонована архітектура досліджувалась у кластері Kubernetes, який складався із двох елементів – головного та робочого вузла. На головному вузлі було встановлено Ubuntu Server 22.04.3 LTS з ядром версії 5.15.0-91-generic, оскільки ця конфігурація забезпечувала найкращу сумісність з Kubernetes v1.34 та підтримку необхідних kernel features для IMA. Робочий вузол отримав Raspberry Pi OS Lite (Debian Bookworm) з модифікованим ядром 6.1.0, до якого було застосовано патч для підтримки шаблону ima-cgpath. До робочого вузла через SPI було підключено апаратний TPM-модуля Infineon OPTIGA SLB 9670.

Процес модифікації ядра Linux для робочого вузла Kubernetes розглядався як необхідна умова коректної роботи механізмів вимірювання цілісності в контейнеризованому середовищі. У стандартній реалізації Linux Integrity Measurement Architecture (IMA) всі вимірювання формуються на рівні вузла і не містять контексту контейнера або Pod'а, у межах якого виконується процес. За відсутності такого контексту неможливо однозначно співвіднести зафіксовані вимірювання з конкретними Pod'ами, що унеможливлює подальшу атестацію та аналіз безпеки на рівні контейнерів у Kubernetes-кластері.

Для усунення цього обмеження було реалізовано модифікацію ядра Linux, спрямовану на розширення структури вимірювань IMA додатковим атрибутом, який відображає приналежність процесу до конкретної контрольної групи. Оскільки в Kubernetes кожен Pod та контейнер ієрархічно представлений у вигляді cgroup, повний шлях cgroup у файловій ієрархії ядра є надійним ідентифікатором виконуваного Pod'а. Саме тому до шаблону вимірювань IMA було додано нове поле cgroup-path, яке зберігає цей шлях у текстовому вигляді. Технічно модифікація була реалізована шляхом внесення змін до файлу security/integrity/ima/ima_template.c, який відповідає за формування та відображення полів шаблону IMA.

Оцінка ефективності розробленої системи верифікації проводилася через порівняння з базовою конфігурацією Kubernetes кластера без механізмів верифікації, а також з reference implementation на базі Keylime. В процесі експериментальних досліджень було проведено заміри часових характеристик, що включали

виконання повного циклу верифікації. Вимірювання виконувалися на ідентичному апаратному забезпеченні Raspberry Pi 3 Model B з 4GB RAM для забезпечення коректного порівняння.

Для Pod'а з двома контейнерами на базі образу Redis розмір IMA log складав приблизно 18000 записів, з яких 247 були релевантні для цього Pod'а. Час атестації одного такого Pod'а становив 4.2 секунди, з розподілом: генерація доказу на робочому вузлі (включаючи TPM quote) – 1.8 секунди, передача по мережі – 0.3 секунди, валідація на модулі верифікації (включаючи парсинг IMA log та запити до підсистеми зберігання білого списку) – 2.1 секунди. При збільшенні кількості одночасних атестацій до 10 Pod'ів середній час зростав до 6.7 секунд за рахунок конкуренції за TPM ресурс, який підтримує лише послідовний доступ.

Порівняння з Keylime для верифікації показало перевагу спеціалізованого рішення. Keylime потребувала 7.3 секунди для верифікації одного робочого вузла (що включало Pod неявно), але не могла розрізняти окремі Pod'и, що робило неможливим вибіркове видалення скомпрометованого Pod'а без перезавантаження вузла. Розроблена система забезпечувала гранулярну верифікацію з додатковими витратами часу лише 4,2 секунди та підтримувала точкове усунення виявлених порушень безпеки на рівні окремих компонентів.

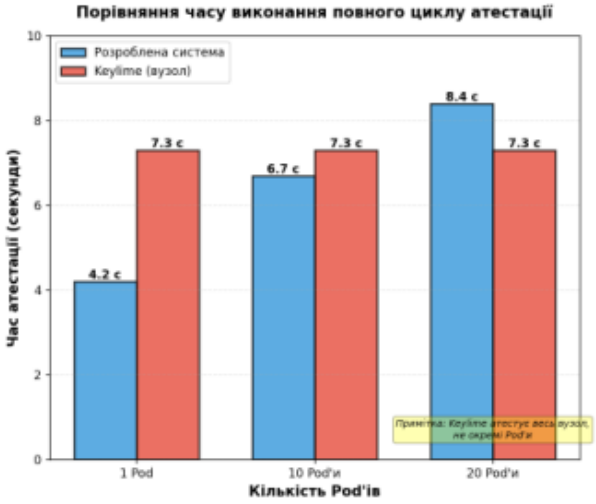


Рисунок 2 – Порівняння часу виконання повного циклу верифікації

Висновки

У межах проведеного дослідження запропоновано архітектуру віддаленої верифікації Kubernetes Pod'ів, яка нативно інтегрується в Kubernetes-кластер через механізми розширюваності платформи без модифікації її ядра. Розроблена система забезпечує гранулярну перевірку цілісності окремих контейнерів на edge-пристроях типу Raspberry Pi, використовуючи апаратний TPM-модуль як корінь довіри та підсистему IMA для збору вимірювань з урахуванням контексту Pod'ів. Ключовими елементами архітектури стали кастомні CRD для представлення станів верифікації, спеціалізовані контролери на Control-Plane для координації процесів та агент на Worker-вузлах для генерації криптографічних доказів, що дозволило реалізувати безперервну атестацію з автоматичною реакцією на виявлені порушення безпеки.

Експериментальна оцінка підтвердила ефективність запропонованого рішення: час повного циклу верифікації одного Pod'а склав 4,2 секунди з можливістю точкового усунення загроз на рівні окремого контейнера, що суттєво перевищує можливості node-level інструментів на кшталт Keylime (7,3 секунди без гранулярності). Отримані результати демонструють практичну придатність архітектури для реальних edge-сценаріїв з обмеженими ресурсами, забезпечуючи високий рівень безпеки без значного впливу на продуктивність кластера та відкриваючи перспективи для подальшого розвитку в напрямку zero-trust моделей для розподілених систем.

Література

1. Keylime: remote boot attestation and runtime integrity management solution. Cloud Native Computing Foundation (CNCF). URL: https://keylime.dev/

2. Zaritto F., Bravi E., Sisinni S., Lioy A. Extending Kubernetes for Pods Integrity Verification. Journal of Network and Systems Management. 2026. № 34 (1).

3. Scopelliti G., Amir-Mohammadian S., Csallner C. Trusting the Cloud-Native Edge: Remotely Attested Kubernetes Workers. 2024 33rd International Conference on Computer Communications and Networks (ICCCN), 2024. arxiv.org/abs/2405.10131.

4. Goethals T., De Turck F., Volckaert B. Extending Kubernetes Clusters to Low-Resource Edge Devices Using Virtual Kubelets. IEEE Transactions on Cloud Computing. 2020. DOI: 10.1109/TCC.2020.3033807.

5. Fernandez G. P., Brito A. Secure container orchestration in the cloud: policies and implementation. Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. New York, NY : ACM, 2019. P. 138–145.

6. Zerraza I., Seghir Z. A., Hemam M. An Efficient Lightweight Authentication and Access Control for IoT Edge Devices. Int. J. Saf. Secur. Eng. 2024. № 14. P. 807–813.

7. Lyu M., Ni Z., Chen Q., Li F. Edge-DPSDG: An Edge-based Differential Privacy Protection Model for Smart Healthcare. IEEE Trans. Big Data. 2024. № 11. P. 21–34.

References

1. Keylime: remote boot attestation and runtime integrity management solution. Cloud Native Computing Foundation (CNCF). URL: https://keylime.dev/

2. Zaritto F., Bravi E., Sisinni S., Lioy A. Extending Kubernetes for Pods Integrity Verification. Journal of Network and Systems Management. 2026. № 34 (1).

3. Scopelliti G., Amir-Mohammadian S., Csallner C. Trusting the Cloud-Native Edge: Remotely Attested Kubernetes Workers. 2024 33rd International Conference on Computer Communications and Networks (ICCCN), 2024. arxiv.org/abs/2405.10131.

4. Goethals T., De Turck F., Volckaert B. Extending Kubernetes Clusters to Low-Resource Edge Devices Using Virtual Kubelets. IEEE Transactions on Cloud Computing. 2020. DOI: 10.1109/TCC.2020.3033807.

5. Fernandez G. P., Brito A. Secure container orchestration in the cloud: policies and implementation. Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. New York, NY : ACM, 2019. P. 138–145.

6. Zerraza I., Seghir Z. A., Hemam M. An Efficient Lightweight Authentication and Access Control for IoT Edge Devices. Int. J. Saf. Secur. Eng. 2024. № 14. P. 807–813.

7. Lyu M., Ni Z., Chen Q., Li F. Edge-DPSDG: An Edge-based Differential Privacy Protection Model for Smart Healthcare. IEEE Trans. Big Data. 2024. № 11. P. 21–34.

Надійшла / Paper received : заповнюється редакцією

Надрукована/Printed : заповнюється редакцією

ДОДАТОК Б

(обов'язковий)

Презентація до захисту кваліфікаційної роботи

Архітектура та протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi

Студент групи КІ2м-24-1, **Вадим ГУРСЬКИЙ**

Керівник: к.т.н., доцент **Катерина БЕРЕЗЬКА**

Хмельницький
2026

1

Об'єкт, предмет, мета дослідження

- Об'єктом дослідження є процеси віддаленої верифікації робочих вузлів та окремих Kubernetes Pod'ів у розподілених edge-кластерах на базі Raspberry Pi з використанням апаратного TPM-модуля.
- Предметом дослідження є методи, засоби та фреймворки віддаленої верифікації робочих вузлів та окремих Kubernetes Pod'ів у розподілених edge-кластерах.
- Метою кваліфікаційної роботи магістра є проведення перевірки цілісності на рівні окремих Pod'ів у розподілених кластерах з обмеженими обчислювальними ресурсами, а також зменшення часу проведення процесу верифікації шляхом дослідження та проєктування архітектури й протоколу віддаленої атестації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi з використанням апаратного TPM-модуля Infineon OPTIGA SLB 9670.

2

Наукова новизна

- набув подальшого розвитку протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi, який дозволяє забезпечити гранулярну перевірку цілісності окремих контейнерів у розподілених кластерах з обмеженими ресурсами, і який відрізняється від відомих рішень модифікацією підсистеми IMA ядра Linux для додавання атрибута cgroup-path до вимірювання та інтеграцією із апаратним TPM-модулем типу Infineon OPTIGA SLB 9670 через SPI-інтерфейс, що дозволило зменшити час розгортання системи у порівнянні із відомим інструментом keylime .
- набула подальшого розвитку архітектура віддаленої верифікації Kubernetes pod'ів, яка відрізняється від відомих прямою інтеграцією з апаратним TPM-модулем Infineon OPTIGA SLB 9670 через SPI-інтерфейс на Raspberry Pi, що дозволило використовувати реальний апаратний корінь довіри замість віртуальної емуляції .

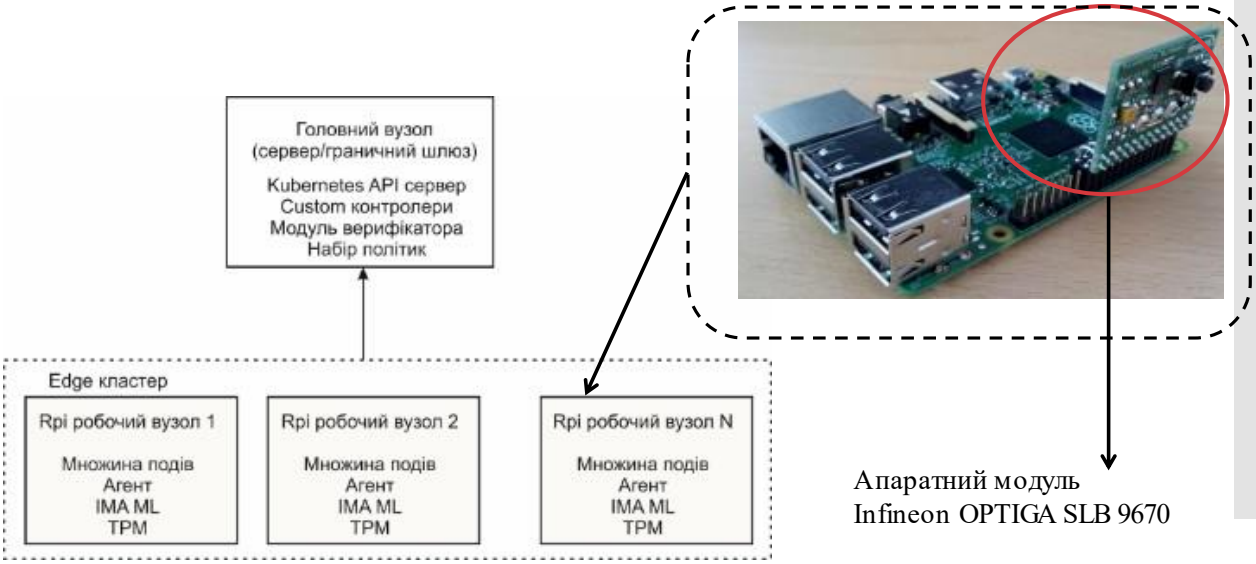
3

Актуальність дослідження

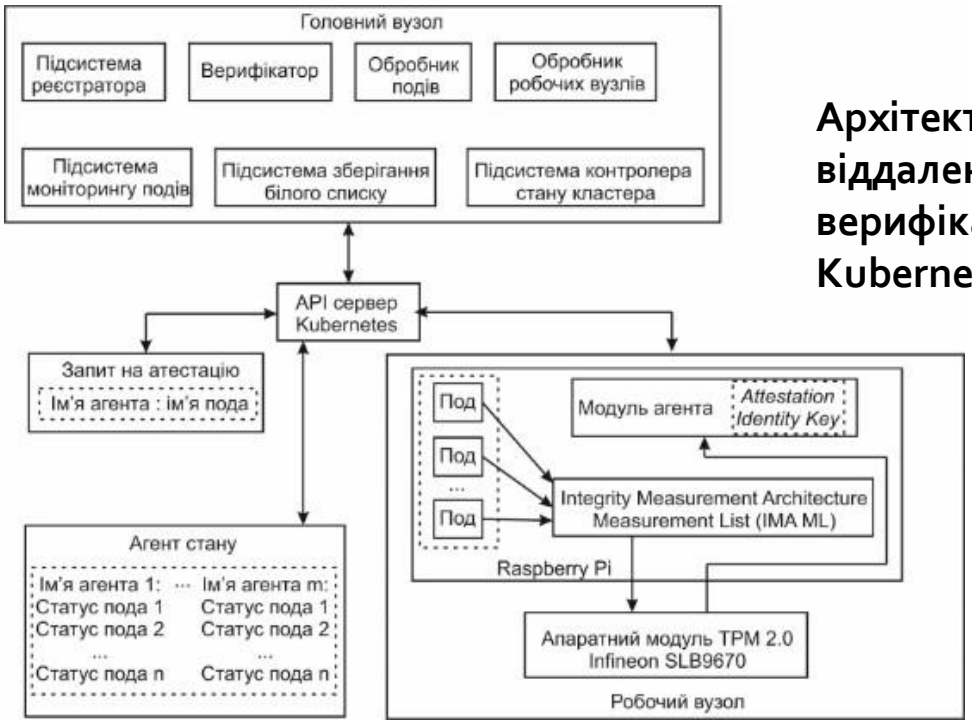
- **Edge computing активно впроваджується** в промислових, транспортних та IoT-системах
- Обчислення **виконуються поблизу джерел даних** , часто у фізично незахищених локаціях
- **Kubernetes став стандартом оркестрації** та використовується навіть на resource-обмежених edge-пристроях (Raspberry Pi)
- **Відсутність нативної віддаленої верифікації в Kubernetes** створює критичну прогалину в безпеці
- Компрометація edge-вузла може призвести до : **витоку конфіденційних даних, маніпуляції результатами обробки, використання інфраструктури для подальших атак**
- Проблема **посилюється в багатоорендному середовищі**, де потрібні криптографічні гарантії довіри до виконання застосунків

4

Організація розподіленого edge-кластеру Kubernetes на основі одноплатних комп'ютерних систем Raspberry Pi

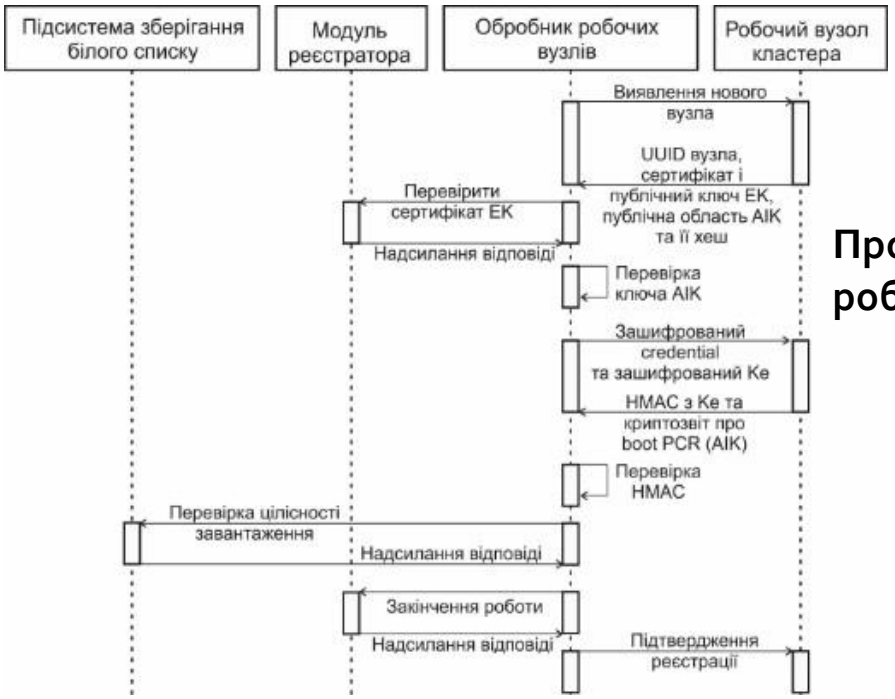


5



Архітектура віддаленої верифікації Kubernetes pod'ів

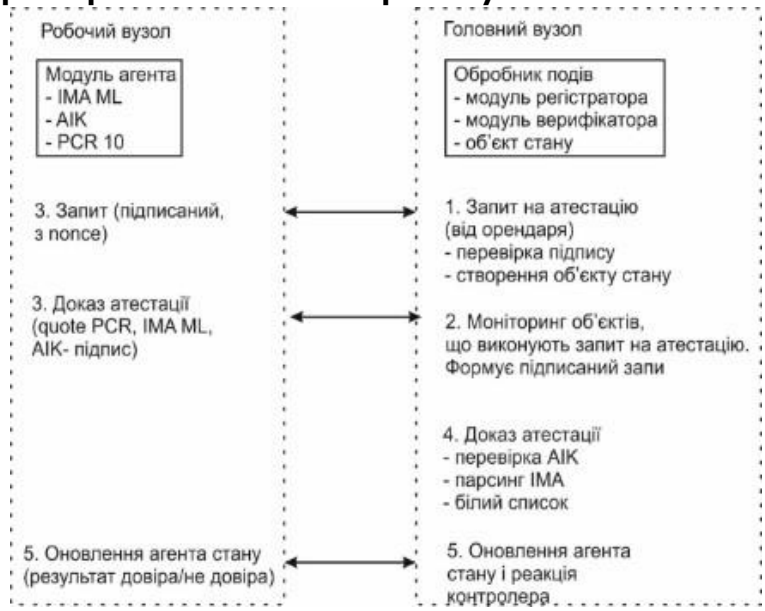
6



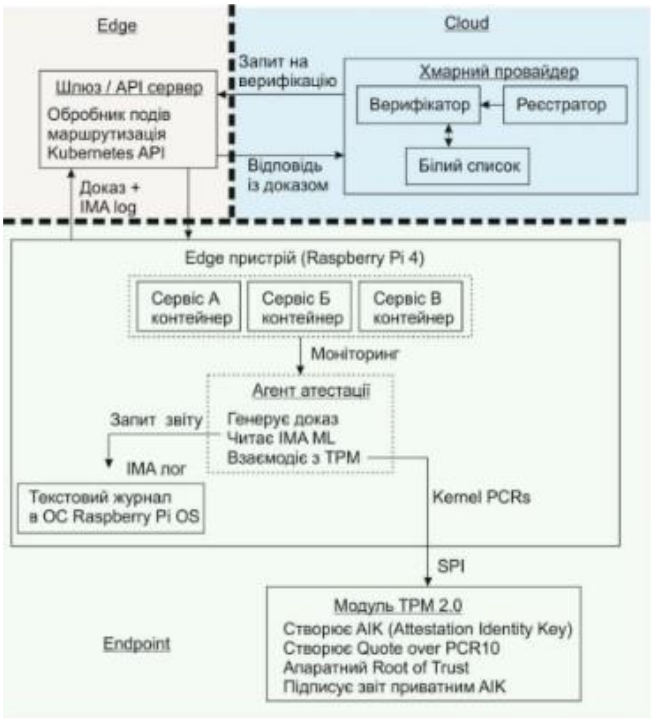
Процес реєстрації робочих вузлів

7

Протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi



8



1. Под запускається на робочому вузлі, що представлений одноплатною комп'ютерною системою Raspberry Pi.
2. Агент на Raspberry Pi:

– читає дані з TPM;
– перевіряє цілісність коду pod'а;

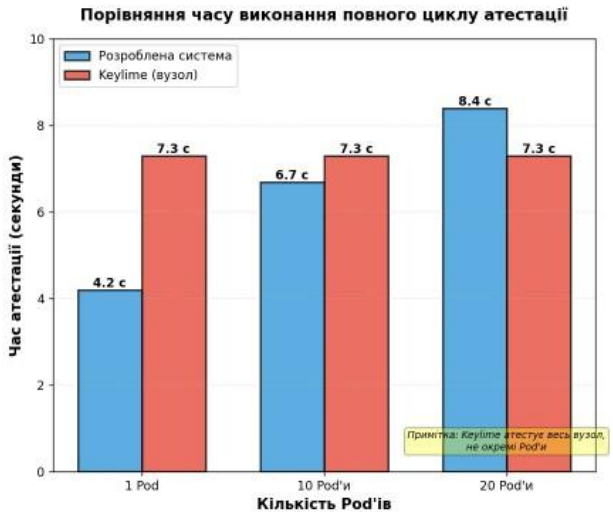
3. Raspberry Pi надсилає доказ до керуючого вузла.
4. На керуючому вузлі:

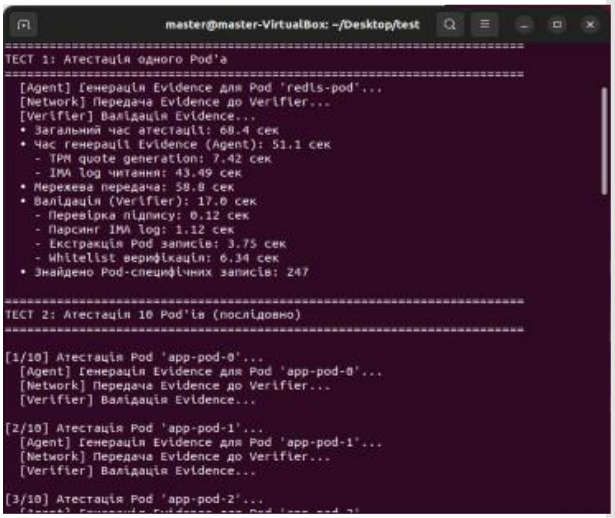
– перевіряється доказ;
– вирішує: довірений под чи не довірений под.

5. Kubernetes кластер виконує реагування, тобто дозволяє або блокує або зупиняє под.

9

Експериментальні дослідження та оцінка ефективності запропонованих рішень





Світлина фрагменту часових замірів

10

Висновки

- У роботі досліджено існуючі методи віддаленої верифікації Kubernetes Pod’ів для edge-пристроїв, з’ясовано їхні обмеження у гранулярності та інтеграції з Kubernetes API, що обґрунтувало розробку спеціалізованої архітектури .
- Запропоновано модель процесу верифікації та архітектуру, яка інтегрується через Kubernetes API, забезпечує ланцюг довіри від TPM на Raspberry Pi до окремого Pod’а та підтримує гранулярну атестацію .
- Розроблений протокол включає ініціацію запиту, формування та перевірку доказів цілісності, а також застосування коригувальних заходів .
- Експериментальна перевірка на edge-кластері підтвердила коректну роботу TPM, модифікованої IMA та гранулярну віддалену верифікацію Pod’ів із часом 4,2 с на Pod, що перевищує швидкодію Keylime і дозволяє точково усувати скомпрометовані застосунки .

Дякую за увагу!

ДОДАТОК В

Реалізація взаємодії з TPM 2.0

Реалізація взаємодії з TPM 2.0 за допомогою бібліотеки go-tpm-tools (на мові Go)

```
func createAttestationKey(tpmDevice io.ReadWriteCloser) (*tpm2.AuthHandle, *tpm2.Public, error) {
   // Отримання Endorsement Primary Key
   ekTemplate := tpm2.Public{
      Type:    tpm2.AlgRSA,
      NameAlg: tpm2.AlgSHA256,
      Attributes: tpm2.FlagFixedTPM | tpm2.FlagFixedParent |
            tpm2.FlagSensitiveDataOrigin | tpm2.FlagAdminWithPolicy |
            tpm2.FlagRestricted | tpm2.FlagDecrypt,
      AuthPolicy: defaultEKAuthPolicy,
      RSAParameters: &tpm2.RSAParams{
         Symmetric: &tpm2.SymScheme{
            Alg:     tpm2.AlgAES,
            KeyBits: 128,
            Mode:    tpm2.AlgCFB,
         },
         KeyBits:  2048,
         Exponent: 0,
      },
   }

   ekHandle, _, err := tpm2.CreatePrimary(
      tpmDevice,
      tpm2.HandleEndorsement,
      tpm2.PCRSelection{},
      "",
      "",
      ekTemplate,
   )
   if err != nil {
      return nil, nil, fmt.Errorf("failed to create EK: %w", err)
   }
   defer tpm2.FlushContext(tpmDevice, ekHandle)

   // Створення AIK як обмеженого підписувального ключа
   aikTemplate := tpm2.Public{
      Type:    tpm2.AlgRSA,
```

```
        NameAlg: tpm2.AlgSHA256,
        Attributes: tpm2.FlagFixedTPM | tpm2.FlagFixedParent |
                tpm2.FlagSensitiveDataOrigin | tpm2.FlagUserWithAuth |
                tpm2.FlagRestricted | tpm2.FlagSign,
        RSAParameters: &tpm2.RSAParams{
            Sign: &tpm2.SigScheme{
                Alg:  tpm2.AlgRSASSA,
                Hash: tpm2.AlgSHA256,
            },
            KeyBits:  2048,
            Exponent: 0,
        },
    }

    aikPrivate, aikPublic, _, _, _, err := tpm2.CreateKey(
        tpmDevice,
        ekHandle,
        tpm2.PCRSelection{},
        "",
        "",
        aikTemplate,
    )
    if err != nil {
        return nil, nil, fmt.Errorf("failed to create AIK: %w", err)
    }

    aikHandle, _, err := tpm2.Load(
        tpmDevice,
        ekHandle,
        "",
        aikPublic,
        aikPrivate,
    )
    if err != nil {
        return nil, nil, fmt.Errorf("failed to load AIK: %w", err)
    }
    return &aikHandle, aikPublic, nil
}
```

ДОДАТОК Г

Реалізація процедури активації креденціалу tpm (TPM2_MAKECREDENTIAL) з використанням EK та AIK

Реалізація процедури активації креденціалу TPM (TPM2_MakeCredential) з використанням EK та AIK.

```
func generateCredentialActivation(ekPub, aikPub *rsa.PublicKey, aikName []byte) ([]byte, []byte, error) {
    // Генерація тимчасового ключа
    ephemeralKey := make([]byte, 32)
    if _, err := rand.Read(ephemeralKey); err != nil {
        return nil, nil, fmt.Errorf("failed to generate ephemeral key: %w", err)
    }

    // Обчислення HMAC від AIK Name
    mac := hmac.New(sha256.New, ephemeralKey)
    mac.Write(aikName)
    credential := mac.Sum(nil)

    // Шифрування credential тимчасовим ключем
    block, err := aes.NewCipher(ephemeralKey[:16])
    if err != nil {
        return nil, nil, fmt.Errorf("failed to create AES cipher: %w", err)
    }

    gcm, err := cipher.NewGCM(block)
    if err != nil {
        return nil, nil, fmt.Errorf("failed to create GCM: %w", err)
    }

    nonce := make([]byte, gcm.NonceSize())
    if _, err := rand.Read(nonce); err != nil {
        return nil, nil, fmt.Errorf("failed to generate nonce: %w", err)
    }

    encryptedCredential := gcm.Seal(nonce, nonce, credential, nil)

    // Шифрування тимчасового ключа публічною частиною EK
    encryptedKey, err := rsa.EncryptOAEP(
        sha256.New(),
        rand.Reader,
```

```
        ekPub,
        ephemeralKey,
        []byte("IDENTITY"),
    )
    if err != nil {
        return nil, nil, fmt.Errorf("failed to encrypt ephemeral key: %w", err)
    }

    return encryptedCredential, encryptedKey, nil
}
```

ДОДАТОК Д

Програмна реалізація процедури генерації атестаційної квоти tpm над PCR-регістрами

```
func generateBootQuote(tpmDevice io.ReadWriteCloser, aikHandle tpmutil.Handle, nonce []byte) ([]byte, error) {
   pcrSelection := tpm2.PCRSelection{
      Hash: tpm2.AlgSHA256,
      PCRs: []int{0, 1, 2, 3, 4, 5, 6, 7, 8, 9},
   }
      quote, signature, err := tpm2.Quote(
      tpmDevice,
      aikHandle,
      "",
      "",
      nonce[:8],
      pcrSelection,
      tpm2.AlgNull,
   )
   if err != nil {
      return nil, fmt.Errorf("failed to generate quote: %w", err)
   }
      // Серіалізація quote та підпису
   quoteData := struct {
      Quote     []byte
      Signature []byte
      PCRs      map[int][]byte
   }{
      Quote:     quote,
      Signature: signature,
      PCRs:      make(map[int][]byte),
   }
      // Зчитування значень PCR
   for _, pcr := range pcrSelection.PCRs {
      pcrValue, err := tpm2.ReadPCR(tpmDevice, pcr, pcrSelection.Hash)
      if err != nil {
         return nil, fmt.Errorf("failed to read PCR %d: %w", pcr, err)
      }
      quoteData.PCRs[pcr] = pcrValue
   }
      return json.Marshal(quoteData)}
```

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Вадим ГУРСЬКИЙ

ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-24-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

10 квітня 2026 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Здобувач: Вадим ГУРСЬКИЙ

Тема: Архітектура та протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи магістра:

Кількість листів креслень – ; кількість сторінок записки 84

1. Короткий зміст роботи та прийнятих рішень У роботі запропоновано архітектуру та протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi

2. Висновок про відповідність роботи дипломному завданню Кваліфікаційна робота магістра відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі досліджено відомі методи та засоби віддаленої верифікації Kubernetes подів для edge-пристроїв, а також проаналізовано їхні функціональні можливості та обмеження. У другому розділі досліджено криптографічні механізми захисту, що ґрунтуються на використанні апаратного модуля довіри для фіксації станів системи та побудови ланцюга довіри. Сформовано теоретичну модель процесу верифікації. У третьому розділі розроблено архітектуру системи, що інтегрується в платформу через стандартні механізми розширюваності, та формалізовано протоколи реєстрації вузлів і верифікації подів. У четвертому розділі проведено експериментальну перевірку запропонованого підходу на мінімальному edge-кластері Kubernetes, розгорнутому на одноплатних комп'ютерах Raspberry Pi з апаратним модулем довіри TPM.

4. Позитивні сторони роботи: Запропонована архітектуру та протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв, що функціонують на базі Raspberry Pi.

5. Негативні сторони роботи: Пропонований підхід хоч і вирішує поставлене завдання, проте створює додаткове навантаження на ресурси edge-пристроїв і контрольної площини, що може негативно впливати на продуктивність системи.

6. Оцінка графічного оформлення та пояснювальної записки роботи: –

7. Відгук про роботу в цілому: В загальному робота виконана на достатньому рівні.

8. Інші зауваження: –

9. Оцінка кваліфікаційної роботи магістра:

Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи магістра вважаю, що робота заслуговує оцінки «добре» 75.00 (С)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) д-р. фіз.-мат. наук, професор, завідувач кафедри ІПЗ Бедратюк Л.П.

"30 квітня" 2026р.

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Вадим ГУРСЬКИЙ

Співавтор:

Назва: Архітектура та протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi

Експерт: Катерина БЕРЕЗЬКА

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:10.28%

Коефіцієнт подібності 2:3.91%

Мікропробіли: 3

Заміна букв: 1

Інтервали: 0

Білі знаки: 6

Дата створення звіту: 2026-04-10 08:51:29.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-04-10 Доцент Андрій Нічепорук

Дата експерт

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Архітектура та протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi
Автор Вадим ГУРСЬКИЙ
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: к.т.н., доцент Катерина БЕРЕЗЬКА

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 10,28% і адресується до 46 першоджерела; та системою Anti-Plagiarism складає 0%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

15.12.2025

Завідувач кафедри _____ Підпис — Ольга ПАВЛОВА (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми _____ — Олег САВЕНКО (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи _____ Підпис — Катерина БЕРЕЗЬКА (Ім'я, ПРІЗВИЩЕ)

Fri Apr 10 10:49:46 EEST 2026, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 0.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 13%**

	Документ		Сумарний збіг по Базі Даних	
ID: 270342 Назва: МКР Архітектура та протокол віддаленої верифікації Kubernetes Pod'ів для edge-пристроїв на базі Raspberry Pi Додано в БД: 2026-04-10 Автора: Вадим ГУРСЬКИЙ Керівники: Катерина БЕРЕЗЬКА Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	114769	713	1575 (1%)	27 (4%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми