Форма №24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій
(повне найменування інституту, назва факультету)

Кафедра інформаційної безпеки та наноелектроніки
(повна назва кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр

(ступінь вищої освіти)

на тему Проєктування та реалізація блокчейн-орієнтованої системи для забезпечення цілісності та автентичності прошивок IoT-пристроїв
(назва теми)

Design and implementation of a blockchain-based system to ensure the integrity and authenticity of IoT device firmware

Виконав: студент 2 курсу,
групи БК-714М
Спеціальності 125 Кібербезпека та захист інформації
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Системи технічного захисту інформації, автоматизація її обробки
БАЙША В.К.
(ПРІЗВИЩЕ та ініціали)

Керівник НЕЛАСА Г.В.
(ПРІЗВИЩЕ та ініціали)
Рецензент САМОЙЛИК С.С.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій
Кафедра інформаційної безпеки та наноелектроніки
Ступінь вищої освіти магістр
Спеціальність 125 Кібербезпека та захист інформації
(код і найменування)
Освітня програма (спеціалізація) Системи технічного захисту інформації, автоматизація її обробки
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІБтаН, к.ф.-м.н., доц.
Андрій КОРОТУН
“____” _____2025 року

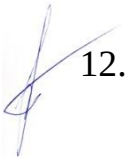
З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

БАЙШИ Вадима Костянтиновича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Проєктування та реалізація блокчейн-орієнтованої системи для забезпечення цілісності та автентичності прошивок IoT пристроїв Design and implementation of a blockchain-based system to ensure the integrity and authenticity of IoT device firmware
Керівник проєкту (роботи) к.т.н., доцент НЕЛАСА Ганна Вікторівна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові,)
затверджені наказом закладу вищої освіти від «_» 2025 року №___
2. Строк подання студентом проєкту (роботи) 18 грудня 2025 року
3. Вихідні дані до проєкту (роботи) Розроблена блокчейн-орієнтована модель забезпечення цілісності прошивок IoT-пристроїв, що включає смарт-контракт, використання криптографічних механізмів SHA-256 та ECDSA.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):архітектура системи, розробка смарт-контракту, реалізація реєстрації та перевірки прошивок; тестування роботи прототипу.
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень): презентація у Microsoft power point (14 слайдів)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
Розділи 1-3	НЕЛАСА Г.В., к.т.н. доцент	 05.09.25	 12.12.25
Нормоконтроль	КОРОЛЬКОВ Р.Ю., к.т.н. доцент		

7. Дата видачі завдання « 05 » вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ пор.	Назва етапів курсового проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1.	Ознайомлення з архітектурою IoT-пристроїв та їх використанням у системах критичної інфраструктури.	*1-2 тиждень*	виконано
2.	Проєктування та розробка смарт-контракту	*3-4 тиждня*	виконано
3.	Розробка та підготовка конфігурації для тестування: формування тестових прошивок, генерація SHA-256 хешів, підготовка ключів ECDSA.	*5 тиждень*	виконано
4.	Деплой смарт-контракту та початкове функціональне тестування.	*6 тиждень*	виконано
5.	Розробка й інтеграція Python-клієнта; тестування реєстрації та перевірки прошивок.	*7-8 тиждень*	виконано
6.	Оформлення ПЗ та графічного матеріалу	*9-10 тиждень*	виконано

Студент(ка)  Вадим БАЙША
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)  Ганна НЕЛАСА
(підпис) (Ім'я ПРІЗВИЩЕ)

АНОТАЦІЯ

Пояснювальна записка до магістерської роботи: 94 с., 5 табл., 25 рис.,5 дод., 20 джерел.

ІОТ, ПРОШИВКА, АВТЕНТИЧНІСТЬ, ЦІЛІСНІСТЬ, БЛОКЧЕЙН, СМАРТ-КОНТРАКТ, ХЕШУВАННЯ, FIRMWARE.

Об'єкт дослідження - процес забезпечення цілісності та автентичності прошивок IoT-пристроїв.

Предмет дослідження: методи, алгоритми та засоби застосування блокчейн-технологій для верифікації firmware в середовищах критичної інфраструктури.

Мета дослідження – спроєктувати та реалізувати блокчейн орієнтовану систему для забезпечення цілістності та автентичності прошивок IoT пристроїв

Результати**:** створено та протестовано діючий смарт-контракт і прототип системи, що забезпечує реєстрацію та перевірку цілісності прошивок IoT-пристроїв.

Рекомендації щодо впровадження: Інтегрувати систему у тестове IoT-середовище та розширити її функціонал під реальні інфраструктурні потреби.

Практична цінність: Підвищує рівень захисту IoT-пристроїв, забезпечуючи контроль автентичності та цілісності їхніх прошивок.

ABSTRACT

Explanatory note to the master's thesis: 94 pages, 5 tables, 25 figures, 5 appendix 20 references.

IOT, FIRMWARE, AUTHENTICITY, INTEGRITY, BLOCKCHAIN, SMART CONTRACT, HASHING, FIRMWARE.

The object of research is the process of ensuring the integrity and authenticity of IoT device firmware.

The subject of the study: is methods, algorithms, and means of applying blockchain technologies for firmware verification in critical infrastructure environments.

The goal of the research: is to design and implement a blockchain-oriented system to ensure the integrity and authenticity of IoT device firmware.

Results: A working smart contract and system prototype have been created and tested, ensuring the registration and verification of the integrity of IoT device firmware.

Recommendations for implementation: Integrate the system into a test IoT environment and expand its functionality to meet real infrastructure needs.

Practical value: Increases the level of protection for IoT devices by ensuring the authenticity and integrity of their firmware.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CID Content Identifier (Ідентифікатор вмісту);

FW Firmware (Прошивка);

HMAC Hash-based Message Authentication Code (код автентифікації повідомлень на основі хешу);

HTTP HyperText Transfer Protocol (Протокол передавання гіпертексту);

IoT Internet of Things (Інтернет речей);

IPFS (InterPlanetary File System) (Міжпланетна файлова система);

ML Machine Learning (Машинне навчання);

MQTT Message Queuing Telemetry Transport (Протокол телеметричного транспорту);

OTA Over-The-Air (Віддалене оновлення прошивки);

RSA Rivest-Shamir-Adleman (Криптографічний алгоритм);

SHA Secure Hash Algorithm (Алгоритм хешування);

TPM Trusted Platform Module (Апаратний модуль довіри);

Wi-Fi Wireless Fidelity (Бездротовий зв’язок);

ABI Application Binary Interface (Двійковий програмний інтерфейс);

AI Artificial Intelligence (Штучний інтелект);

AES Advanced Encryption Standard (Алгоритм симетричного шифрування);

ECDSA Elliptic Curve Digital Signature Algorithm (Алгоритм цифрового підпису на еліптичних кривих);

ECDSA Elliptic Curve Digital Signature Algorithm (Алгоритм цифрового підпису на еліптичних кривих);

PKI Public Key Infrastructure (Інфраструктура відкритих ключів).

ВСТУП

Сучасний розвиток технологій Інтернету речей (Internet of Things, IoT) привів до широкого впровадження «розумних» пристроїв у промисловості, енергетиці, транспорті, медицині та інших сферах критичної інфраструктури. Ці пристрої забезпечують автоматизацію процесів, збір та обробку великих обсягів даних, а також дистанційне керування об’єктами в реальному часі. Разом із тим зростає залежність таких систем від надійності та безпеки програмного забезпечення, яке встановлюється у вигляді прошивок (firmware). Будь-яка компрометація або несанкціоноване оновлення прошивки може призвести до порушення роботи критичних систем, витоку конфіденційних даних чи навіть до техногенних аварій.

Традиційні централізовані моделі перевірки автентичності прошивок не завжди здатні забезпечити достатній рівень довіри, оскільки компрометація центрального сервера або втручання у ланцюг постачання (Supply Chain Attack) можуть призвести до встановлення шкідливого ПЗ.

Одним із перспективних напрямів вирішення цієї проблеми є використання блокчейн-технологій, що забезпечують незмінність записів, прозорість операцій а також розподілений механізм довіри. Завдяки цим властивостям блокчейн може бути використаний для створення децентралізованої системи реєстрації та перевірки прошивок IoT-пристроїв, у якій кожне оновлення фіксується у вигляді транзакції, а автентичність firmware підтверджується криптографічними методами.

1 ТЕОРЕТИЧНІ ОСНОВИ ЗАБЕЗПЕЧЕННЯ ЦІЛІСНОСТІ ТА АВТЕНТИЧНОСТІ ПРОШИВОК IoT-ПРИСТРОЇВ

1.1 Архітектура IoT-систем та їх роль у сучасних розподілених середовищах

Інтернет речей (Internet of Things, скорочено IoT) - це глобальна мережа підключених до Інтернету речей - пристроїв, оснащених сенсорами, датчиками, засобами передавання сигналів. [1]. Ідея є в тому, щоб зробити об’єкти «розумними» й пов’язаними між собою, створюючи єдиний інтегрований простір для збирання та оброблення інформації, що є актуальним для поліпшення якості життя, оптимізації процесів і підвищення ефективності. Таким чином, інтернет речей - це пристрої, об’єднані в одну бездротову мережу для вирішення конкретного завдання [2].

Типова архітектура IoT-системи включає три основні рівні:

- рівень пристроїв (Device Layer) - сенсори, контролери, мікроконтролери та інші «розумні» пристрої, що збирають дані з навколишнього середовища;

- рівень мережевої взаємодії (Network Layer) - канали передачі даних, що забезпечують комунікацію між пристроями, шлюзами та хмарними сервісами (наприклад, через Wi-Fi, Bluetooth, MQTT або HTTP);

- рівень застосунків (Application Layer) - програмні платформи, аналітичні сервіси або системи керування, які приймають дані від пристроїв і забезпечують ухвалення рішень.

У більшості сучасних IoT-рішень використовується хмарна або гібридна архітектура, де дані з пристроїв надходять у хмарні сервіси для централізованої обробки, аналізу та зберігання потрібної інформації. Однак така централізація створює певні ризики, пов’язані з надійністю каналів зв’язку, затримками при обробці даних і можливістю несанкціонованого доступу до конфіденційної інформації користувача. Тому все більшого поширення набувають розподілені

архітектури (edge та fog computing), у яких частина обчислень виконується безпосередньо на периферії мережі - ближче до пристроїв.

IoT-системи активно інтегруються з технологіями штучного інтелекту (AI), машинного навчання (ML) та блокчейну, що дозволяє підвищити рівень автономності, безпеки та достовірності даних. Зокрема, використання блокчейн-технологій у поєднанні з IoT відкриває нові можливості для забезпечення цілісності інформації, автентичності пристроїв і довіри між різними компонентами розподіленого середовища.

IoT виступає фундаментальною технологією цифрової трансформації, забезпечуючи взаємодію між фізичними об’єктами та цифровими системами. Його роль у розподілених середовищах полягає у формуванні інфраструктури, де дані з численних пристроїв можуть оброблятися, передаватися та перевірятися у безпечний і надійний спосіб, що є основою для подальшого впровадження механізмів контролю цілісності firmware та підвищення рівня безпеки IoT-рішень.

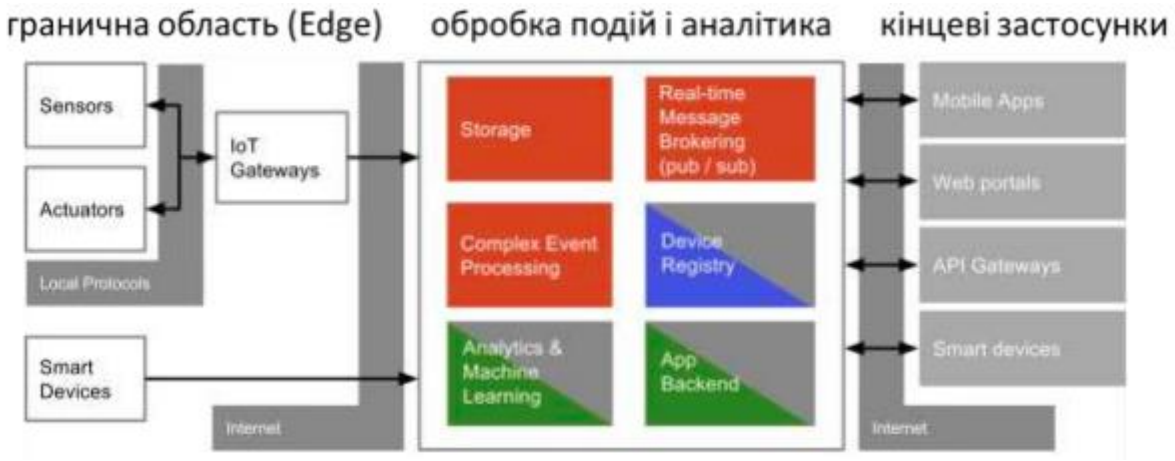


Рисунок 1.1 – Архітектура IoT [3].

1.2 Загрози безпеці IoT-пристроїв, пов’язані з компрометацією прошивок

Одним із ключових елементів безпеки IoT-пристроїв є цілісність і автентичність їх програмного забезпечення, зокрема вбудованих прошивок Прошивка визначає логіку роботи пристрою, реалізує мережеві протоколи, шифрування, обробку даних та інші функції. Будь-яка несанкціонована зміна firmware може призвести до порушення працездатності пристрою, витоку даних або створення умов для масштабної кібератаки.

Компрометація прошивки - це коли програмний код пристрою змінюється навмисно або випадково, унаслідок чого він починає виконувати операції, які не відповідають закладеній виробником логіці роботи. Такі зміни можуть відбутися під час оновлення, на етапах розробки або під час передачі firmware до користувача. Подібні атаки часто використовуються для інтеграції шкідливих компонентів, прихованих механізмів доступу або з метою отримання несанкціонованого управління пристроєм.

Основні загрози, пов’язані з компрометацією прошивок:

- підміна firmware під час оновлення;
- несанкціоноване оновлення з недостовірного джерела;
- використання застарілої або вразливої версії firmware;
- інжекція шкідливого коду у процесі виробництва;
- маніпуляції з процесом перевірки цифрового підпису;
- зміна конфігураційних параметрів або завантажувача.

Компрометація прошивки є особливо небезпечною, оскільки вона дозволяє зловмиснику отримати низькорівневий доступ до системи - тобто контроль на рівні апаратного забезпечення. Такі атаки часто залишаються непоміченими стандартними антивірусними або мережевими засобами захисту, адже firmware виконується поза межами основної операційної системи.

Для запобігання цього необхідно впроваджувати комплексні механізми безпеки:

- цифрові підписи для перевірки автентичності firmware, криптографічні хеші для контролю цілісності, безпечні протоколи передачі даних, а також багаторівневу систему автентифікації джерел оновлень;

- застосування децентралізованих технологій, зокрема блокчейну, дозволяє створити розподілений реєстр автентичних прошивок, що унеможливлює їх несанкціоновану підміну та підвищує рівень довіри до процесу оновлення IoT-пристроїв.

У таблиці 1.1 наведено основні типи загроз, пов’язаних із компрометацією прошивок IoT-пристроїв, та відповідні методи їх нейтралізації.

Таблиця 1.1 – Типові загрози компрометації firmware та способи протидії

№	Тип загрози	Опис загрози	Основні методи протидії
1	2	3	4
1	Підміна прошивки під час оновлення	Зловмисник перехоплює або змінює файл firmware, підміняючи його шкідливою версією	Використання цифрових підписів, перевірка хеш-сум, захищені канали передачі
2	Несакціоноване оновлення з недостовірного джерела	Пристрій отримує firmware не від офіційного виробника	Автентифікація джерела оновлення, перевірка сертифікатів, список довірених ключів
3	Встановлення застарілої або вразливої версії firmware	Зловмисник навмисно змінює версію на старішу з відомими вразливостями	Контроль версій firmware, заборона встановлення старіших версій, перевірка метаданих

Кінець таблиці 1.1

1	2	3	4
4	Компрометація ланцюга постачання	Шкідливий код інтегрується у firmware ще на етапі розробки	Перевірка підписів виробника, аудит програмного ланцюга, контроль на етапах розробки.
5	Маніпуляція з перевіркою цифрового підпису	Модифікація алгоритму перевірки підпису або обхід перевірки цілістності	Застосування апаратно захищених модулів, перевірка підписів у середовищі
6	Зміна завантажувача або параметрів ініціалізації	Порушення механізму завантаження для запуску неавторизованого firmware	Використання захищених bootloader-ів, апаратне шифрування секцій пам’яті, контроль завантаження

Узагальнюючи наведені вище типи загроз та відповідні методи протидії, можна стверджувати, що питання захисту прошивок IoT-пристроїв є багаторівневим і охоплює як криптографічні механізми, так і організаційні та архітектурні заходи. Лише комплексний підхід, який включає перевірку цілісності, автентичності, контроль ланцюга постачання, захищене завантаження та аудит оновлень, здатен гарантовано запобігти компрометації firmware і забезпечити стійку роботу IoT-інфраструктури навіть за умов активної кіберзагрози. У подальших розділах ці принципи будуть покладені в основу розроблення та обґрунтування блокчейн-орієнтованої системи контролю прошивок.

1.3 Методи забезпечення цілісності firmware

Забезпечення цілісності firmware передбачає гарантування того, що оригінальний образ прошивки не був змінений від часу його створення виробником до моменту встановлення на пристрій. Для цієї мети застосовуються криптографічні та некриптографічні методи контролю цілісності, серед яких основними є хешування, контрольні суми та цифрові підписи.

Криптографічні хеш-функції перетворюють довільний за розміром вхідний файл на фіксований короткий рядок. Навмисна зміна будь-якого біта у файлі призведе до радикальної зміни значення хешу, що робить хешування ефективним інструментом виявлення модифікацій. Хеші використовуються як «відбитки» прошивок: виробник обчислює хеш образу та передає його разом із метаданими у довірений реєстр або зберігає у підписаному пакеті оновлення. Серед обмежень це є нездатність хеш-функції самостійно підтвердити походження файлу, знання хешу дозволяє лише виявити зміну, але не встановити, хто її здійснив.

Некриптографічні контрольні суми, наприклад, CRC32 забезпечують просту і швидку перевірку помилок передачі даних або помилкових змін. Вони ефективні проти випадкових помилок, але вразливі до навмисних змін: атаку, що модифікує файл узгоджено з обчисленою сумою, здійснити набагато простіше, ніж злом поведінки криптографічного хешу. Тому контрольні суми часто використовуються як додатковий, але не єдиний метод контролю цілісності.

Щоб гарантувати високий рівень цілісністості, так і автентичністі firmware, застосовуються схеми цифрового підпису RSA, ECDSA. Виробник підписує хеш приватним ключем, пристрій перевіряє підпис публічним ключем, закладеним у захищеному сховищі. Цифрові підписи забезпечують коректність походження оновлення - навіть маючи доступ до підписаного файлу і його

хешу, зловмисник не зможе створити коректний підпис без приватного ключа. Для підвищення надійності підписи використовують апаратні модулі зберігання ключів TPM, Secure Element та механізми Secure Boot, які перевіряють ланцюжок довіри від завантажувача до основної прошивки.

Важливими практичними аспектами є:

- управління ключами, впровадження механізмів ротації й відкликання ключів;

- захист метаданих оновлення, для запобігання rollback-атак;

- забезпечення безпечного каналу доставки оновлення, для зниження ризику перехоплення в мережі.

У контексті блокчейн-орієнтованої системи хеші прошивок та метадані можуть зберігатися в незмінному реєстрі, тобто у смарт - контракті. Це дозволяє пристрою не лише перевірити підпис виробника локально, а й зіставити хеш із тим, що зареєстровано у блокчейні - таким чином посилюється захист від підміни файлу в каналі доставки або від компрометації локального сховища метаданих. Водночас, підписування та управління ключами залишаються критично важливими, блокчейн фіксує запис, але не замінює криптографічні механізми автентифікації.

У таблиці 1.2 наведено порівняльну характеристику основних методів забезпечення цілісності firmware, які використовуються для захисту IoT-пристроїв від несанкціонованих змін.

Таблиця 1.2 – Порівняльна характеристика методів забезпечення цілісності firmware

№	Метод	Призначення	Переваги	Недоліки	Захист
1	Контрольні суми (CRC, Checksum)	Виявлення випадкових помилок передачі або запису даних	Простота реалізації, висока швидкість обчислення	Не захищає від навмисних змін, легко підробити	Низький
2	Криптографічне хешування	Виявлення будь-яких змін у файлі, контроль цілістності firmware	Неможливість відновлення вихідних даних, чутливість до змін	Не підтверджує джерело походження, лише фіксує факт зміни	Середній
3	Цифрові підписи RSA, ECDSA	Забезпечення цілістності та автентичності firmware	підтвердження джерела оновлення, запобігання підміни	Складність управління ключами, потреба у безпечному сховищі	Високий

Як видно з таблиці, контрольні суми придатні лише для виявлення випадкових помилок, тоді як криптографічні хеші та цифрові підписи забезпечують значно вищий рівень захисту. Поєднання цих методів дозволяє створити надійний механізм контролю цілісності firmware.

1.4 Методи автентифікації та забезпечення довіри до джерел оновлень

Автентифікація джерела оновлення firmware є одним із ключових аспектів забезпечення безпеки IoT-пристроїв. Вона покликана гарантувати, що отримане оновлення надходить саме від офіційного виробника, не містить несанкціонованих змін і може бути безпечно встановлене на пристрій. За відсутності надійних механізмів автентифікації існує ризик, що зловмисник зможе підмінити прошивку або внести до неї шкідливі модифікації, що призведе до компрометації всієї системи.

Мета автентифікації полягає у встановленні довіри між пристроєм і джерелом оновлення. Для цього використовуються різні криптографічні та протокольні механізми - цифрові підписи, сертифікати, токени доступу або навіть децентралізовані системи реєстрації. Найбільш простим способом є застосування статичних публічних ключів або сертифікатів виробника, які вбудовуються в пристрій під час виробництва. Під час отримання оновлення пристрій перевіряє цифровий підпис firmware, використовуючи цей ключ. Якщо підпис є дійсним, оновлення вважається автентичним. Такий підхід забезпечує базовий рівень довіри, але має обмеження - у разі компрометації ключа його складно замінити, а в масштабних IoT-мережах оновлення великої кількості пристроїв стає проблематичним.

Для складніших систем використовується ієрархічна інфраструктура відкритих ключів, у межах якої сертифікаційні центри видають сертифікати виробникам або серверам оновлень. Пристрій, маючи у своєму сховищі публічний ключ, може перевірити справжність сертифіката та, відповідно, підпис firmware. Такий механізм забезпечує централізоване управління довірою та є масштабованим, проте залежить від безпеки самого центру сертифікації. У разі його компрометації може бути порушено довіру до всієї інфраструктури, тому використовуються допоміжні механізми - списки відкликаних сертифікатів.

Одним із поширеним підходом є використання токенів автентифікації API-ключів або протоколів OAuth. Токени дозволяють контролювати доступ пристроїв до серверів оновлень, обмежуючи права на завантаження firmware. Вони забезпечують гнучкість і зручність керування доступом, проте потребують захищеного каналу зв’язку і надійного зберігання секретів, оскільки витік токенів може призвести до несанкціонованого доступу.

Перспективним напрямом є впровадження децентралізованих методів перевірки автентичності на основі блокчейн-технологій. У цьому випадку замість централізованих серверів чи сертифікаційних центрів використовується розподілений реєстр, у якому зберігаються хеші прошивок та їх метадані. IoT-пристрій може перевіряти firmware, зіставляючи її хеш із записом у блокчейні, що виключає можливість підміни чи видалення даних. Такий підхід забезпечує прозорість і незмінність процесу оновлення, усуває залежність від центрального органу довіри, але потребує додаткових обчислювальних ресурсів і підключення до блокчейн-мережі [4].

У таблиці 1.3 наведено порівняльну характеристику основних методів автентифікації джерел оновлень firmware, що використовуються для забезпечення довіри між виробником та пристроєм.

Таблиця 1.3 – Порівняльна характеристика методів автентифікації джерел оновлень firmware

№	Метод	Суть методу	Переваги	Недоліки
1	Статичні ключі або сертифікати виробника	Перевірка джерела firmware за заздалегідь встановленим сертифікатом	Простота реалізації, незалежність від зовнішніх сервісів	Потребує оновлення ключів у разі компрометації, складність масштабування

Кінець таблиці 1.3

1	2	3	3	.4
2	Ієрархічна модель сертифікації	Використання ієрархії центрів сертифікації для перевірки справжності оновлень	Високий рівень довіри, централізоване управління, масштабованість	Залежність від CA, ризик компрометації
3	Токени автентифікації	Перевірка права доступу до сервера оновлень через унікальні токени	Динамічність, зручність керування правами.	Потребує безпечного каналу зв’язку, ризик витоку токенів

Можна зазначити, що ефективна система автентифікації оновлень повинна поєднувати кілька методів - криптографічну перевірку цифрового підпису firmware, верифікацію сертифікатів або токенів доступу та збереження відомостей про автентичні оновлення у незмінному реєстрі. Поєднання класичних криптографічних підходів із блокчейн-технологіями дозволяє створити стійкий до атак механізм довіри між виробником і пристроєм, який забезпечує безпечний процес оновлення firmware у масштабних IoT-середовищах.

1.5 Застосування блокчейн-технологій у сфері IoT-безпеки

Інтернет речей об’єднує величезну кількість пристроїв, які взаємодіють між собою та з серверними платформами через мережу. У таких системах надзвичайно важливо гарантувати достовірність даних, цілісність програмного забезпечення та контроль автентичності оновлень. Одним із сучасних напрямів вирішення цих завдань є використання блокчейн-технологій.

Блокчейн - це технологія зберігання даних у вигляді ланцюжка блоків, де кожен запис підтверджується мережею учасників і не може бути змінений заднім числом [5].

Включення блокчейнів в екосистеми IoT дозволяє створити децентралізовану модель довіри, яка підтримує безпечну однорангову комунікацію між пристроями без посередників. Незмінні функції блокчейнів щодо ведення обліку та смарт-контрактів ще більше посилюють автоматизацію та безпеку IoT, гарантуючи, що дані залишаються незмінними після їх запису. Інтеграція блокчейнів з IoT підвищує прозорість і підзвітність завдяки встановленим аудиторським слідовим і перевіреним транзакціям, що є критично важливим для управління ланцюгами поставок, розумних міст, охорони здоров'я та промислової автоматизації [6].

Для реалізації подібних систем використовуються смарт-контракти, які автоматизують процес перевірки. Смарт-контракт може забезпечувати функції реєстрації нових версій firmware, перевірки відповідності хешів та збереження історії оновлень. При цьому сам блокчейн виступає як незмінне джерело даних, що гарантує цілісність і прозорість процесу.

Однією з популярних платформ для створення таких рішень є Ethereum, яка підтримує смарт-контракти, написані мовою Solidity. У межах цієї роботи буде реалізовано прототип системи, що демонструє принципи фіксації та перевірки firmware за допомогою смарт-контракту. Система передбачає

можливість реєстрації хешу прошивки виробником і подальшої перевірки цього хешу перед оновленням IoT-пристрою.

Застосування блокчейн-технологій у цій сфері дозволяє підвищити рівень довіри між учасниками системи оновлень, зменшити ризик підміни firmware та забезпечити прозорий механізм контролю версій. Попри те, що повна інтеграція блокчейну у промислові IoT-системи може потребувати значних ресурсів, навіть часткове використання цієї технології, наприклад, для верифікації хешів прошивок уже суттєво підвищує рівень безпеки процесів оновлення.

1.6 Аналіз сучасних рішень і підходів

Питання забезпечення безпеки та довіри в системах Інтернету речей активно досліджується провідними виробниками програмного та апаратного забезпечення. На ринку існує низка рішень, спрямованих на перевірку автентичності, цілісності та контроль оновлень програмного забезпечення IoT-пристроїв. Їхні підходи різняться за рівнем централізації, використаними технологіями, типами сертифікації та способами управління ключами.

Одним із відомих корпоративних рішень є Cisco IoT Security, яке реалізує багаторівневий підхід до захисту мережевої інфраструктури та пристроїв. Cisco пропонує механізми перевірки достовірності пристроїв, моніторинг аномальної активності та централізоване управління оновленнями. Основу захисту становить система цифрових сертифікатів та автентифікація пристроїв.

Firmware пакети підписуються виробником, а їх перевірка здійснюється на етапі встановлення. Таке рішення орієнтоване переважно на корпоративні мережі з великою кількістю вузлів, де важлива централізована політика безпеки. Платформа Microsoft Azure IoT пропонує власну екосистему керування пристроями Azure IoT Hub, яка включає функції безпечної реєстрації, автентифікації та оновлення firmware. Кожен пристрій отримує

унікальні облікові дані, а обмін даними здійснюється через захищені канали. Для перевірки цілісності використовується цифровий підпис firmware та хеш-функції. Крім того, Azure підтримує інтеграцію з хмарними системами моніторингу, що дозволяє оперативно виявляти невідповідності у версіях програмного забезпечення.

Компанія ARM розробила платформу Pelion Device Management, яка забезпечує повний цикл управління пристроями - від виробництва до оновлення firmware. У межах цієї платформи реалізовано автентифікацію пристроїв на основі сертифікатів X.509, перевірку цифрових підписів firmware та контроль цілісності на етапі завантаження. Pelion дозволяє автоматично перевіряти відповідність firmware стандартам безпеки та використовувати віддалене оновлення через зашифровані канали. Це рішення орієнтоване на виробників обладнання, які прагнуть централізовано контролювати процес оновлень.

Також одним із напрямом є використання блокчейн-технологій у сфері IoT-безпеки. Одним із прикладів є проєкт IOTA, який застосовує розподілену книгу транзакцій для забезпечення достовірності даних між IoT-пристроями. IOTA орієнтована на мікротранзакції та обмін даними між пристроями без участі посередників. Платформа не є класичним блокчейном, вона ілюструє можливість використання розподілених технологій для підтвердження достовірності інформації у масштабних IoT-мережах.

Аналіз наведених рішень показує, що сучасні платформи безпеки IoT використовують переважно централізовані моделі перевірки та управління довірою, засновані на інфраструктурі відкритих ключів. У той же час поступово зростає інтерес до інтеграції розподілених технологій зберігання даних, які забезпечують додатковий рівень контролю та прозорості. Це свідчить про тенденцію переходу від повністю централізованих до гібридних рішень, де блокчейн може виступати допоміжним компонентом у процесі підтвердження автентичності firmware.

Таким чином, огляд сучасних рішень свідчить, що використання блокчейн-технологій у поєднанні з класичними криптографічними методами є

перспективним напрямом для підвищення безпеки процесів оновлення та управління firmware у системах Інтернету речей.

1.7 Криптографічні алгоритми, що використовуються для захисту firmware

Криптографія відіграє ключову роль у захисті прошивок IoT-пристроїв, адже саме вона дозволяє гарантувати, що файл не було змінено та що він дійсно походить від виробника. Для цього застосовуються різні методи - хешування, цифрові підписи, шифрування та механізми безпечного завантаження.

Найпоширеніший спосіб перевірки цілісності firmware - це використання криптографічних хеш-функцій, таких як SHA-256 або SHA-3. Хешування дозволяє отримати коротке унікальне значення, яке відповідає вмісту файлу. Якщо хоч один байт у прошивці буде змінено, її хеш одразу стане іншим. Виробник зазвичай обчислює хеш прошивки після компіляції й зберігає його у базі або додає до метаданих, а пристрій під час оновлення перевіряє, чи збігається отримане значення з оригінальним. Так можна швидко виявити будь-які зміни у файлі.

Щоб не лише виявляти зміну, а й бути впевненим, що прошивка дійсно походить від офіційного джерела, використовуються цифрові підписи. Це означає, що виробник підписує firmware своїм приватним ключем, а пристрій має відповідний публічний ключ і перевіряє підпис перед встановленням. Якщо підпис не збігається - файл вважається підозрілим. Найпоширеніші алгоритми підпису - RSA, ECDSA та Ed25519. У сучасних системах частіше застосовують ECDSA, бо він забезпечує високий рівень безпеки при невеликих розмірах ключів і швидшій роботі, що особливо важливо для пристроїв із обмеженими ресурсами.

Для захищеної передачі прошивки через мережу можуть використовуватися симетричні алгоритми шифрування, такі як AES. Зазвичай

це потрібно тоді, коли firmware містить закриту інформацію або передається через відкриті канали зв’язку. Якщо ж прошивка є публічною, то шифрування не обов’язкове - достатньо перевірки підпису та хешу.

У більш простих системах для контролю автентичності може використовуватися HMAC (Hash-based Message Authentication Code). Це коли пристрій і сервер мають спільний секретний ключ, за допомогою якого перевіряється, чи дані не були змінені. Однак цей метод підходить лише тоді, коли можна безпечно зберігати спільний ключ - у більш складних системах безпечніше застосовувати цифрові підписи.

Важливою частиною системи безпеки є також Secure boot - механізм, який перевіряє цифрові підписи всіх компонентів при запуску пристрою. Це створює так званий «ланцюжок довіри», апаратна частина пристрою перевіряє підпис завантажувача, а завантажувач - підпис самої прошивки. Якщо хоча б один етап перевірки не проходить, пристрій не запускається або переходить у режим відновлення. Такий підхід захищає від запуску неавторизованого коду навіть у випадку фізичного доступу до пристрою.

Ключі, які використовуються для підписів і перевірок, мають бути добре захищені. Для цього часто застосовують спеціальні апаратні модулі, Secure Element або Enclave, які зберігають приватні ключі й виконують криптографічні операції всередині себе, не дозволяючи витягти ключ назовні. Також важливо мати можливість оновлювати або відкликати ключі, якщо вони скомпрометовані. У централізованих системах це робиться через сертифікати й списки відкликаних ключів, а у розподілених - через запис статусу в блокчейні або інші реєстри.

Для забезпечення автентичності прошивок в IoT-системах критичним є вибір алгоритму цифрового підпису. У рисунку нижче представлено порівняння продуктивності RSA та ECDSA.

	Створення підпису	Перевірка підпису
RSA(1024 bit)	25мс	<2мс
ECDSA(160 bit)	32мс	33мс
RSA(2048 bit)	120мс	5мс
ECDSA(216 bit)	68мс	70мс

Рисунок 1.2 – Порівняння продуктивності RSA та ECDSA [7]

Як видно, ECDSA має суттєві переваги при використанні в пристроях із обмеженою обчислювальною потужністю, що робить його більш доцільним вибором для IoT-середовища

2 ПРОЄКТУВАННЯ БЛОКЧЕЙН-ОРІЄНТОВАНОЇ СИСТЕМИ ПЕРЕВІРКИ FIRMWARE IOT-ПРИСТРОЇВ

2.1 Визначення вимог до системи безпеки firmware

При розробці системи безпеки для прошивок IoT-пристроїв основним завданням є забезпечити перевірку справжності firmware до її встановлення. Прошивка є одним із найвразливіших компонентів у пристрої, тому її цілісність і автентичність мають перевірятись ще до початку оновлення. Щоб реалізувати цю перевірку без надмірного навантаження на пристрій, вся логіка перевірки була винесена за межі IoT - і зосереджена на стороні користувача або сервісу-посередника. Файл прошивки перед публікацією проходить хешування, обчислюється контрольна сума за алгоритмом SHA-256. Це дозволяє фіксувати стан файлу на момент публікації. Крім цього, виробник підписує цей хеш з використанням алгоритму ECDSA. Публічний ключ, яким перевіряється підпис, заздалегідь відомий системі й може бути прив’язаний до Ethereum-адреси виробника.

Щоб зробити перевірку на високому рівні безпеки, усі мета-дані - включно з хешем, назвою прошивки, версією, підписом, адресою виробника, часом публікації та статусом, активно або відкликана, післлля чого записуються у смарт-контракт, розгорнутий на приватному блокчейні Ethereum. Такі дані неможливо змінити після публікації, і це виключає можливість несанкціонованого редагування. Якщо якась прошивка пізніше виявиться вразливою, її можна відмітити як відкликану - цей статус також перевіряється перед встановленням. Щоб уникнути ручної взаємодії з блокчейном, розроблений інтерфейс, який дозволяє просто обрати файл - а вся перевірка відбувається автоматично. Уся система функціонує на базі Ganache - це локальне середовище для тестування Ethereum-смарт-контрактів, що дозволяє безпечно перевірити логіку роботи без витрат і зовнішніх залежностей.

Таким чином, система орієнтована на реальні потреби безпеки, враховує обмеження IoT-пристроїв і забезпечує надійну перевірку firmware до її встановлення, без складної взаємодії з боку користувача.

2.2 Модель загроз і критерії безпеки

При проєктуванні системи перевірки прошивок IoT-пристроїв необхідно враховувати можливі загрози, які виникають не лише в самих пристроях, а й у логіці перевірки, передачі та зберігання firmware. Навіть якщо сам файл прошивки створено коректно, зловмисник може втрутитися в процес на інших етапах - наприклад, перехопити її при передачі, модифікувати файл, видати підроблений варіант за офіційний або змусити пристрій встановити старішу версію з відомими вразливостями.

Модель загроз, яка використовується в даній роботі, ґрунтується на характеристиках запропонованої архітектури: централізований виробник, децентралізоване зберігання хешів firmware у блокчейні, клієнтський інтерфейс для перевірки, а також симуляція IoT-пристрою, який виконує локальну перевірку хешу. Це дозволяє сфокусуватись саме на тих точках взаємодії, де можуть з’явитися вразливості.

Варто враховувати, що в реальних умовах IoT-пристрої часто мають обмежені ресурси, а процес оновлення прошивки може відбуватись віддалено. Саме тому перевірка автентичності та цілісності firmware має відбуватись заздалегідь або автоматично, з мінімальним навантаженням на пристрій. Щоб гарантувати безпеку всього ланцюга - від публікації до встановлення - необхідно виявити потенційні вектори атак, сформулювати критерії безпеки та закласти їх у основу проєктної частини.

В таблиці нижче наведено типові сценарії атак з коротким описом ризиків, які можуть виникнути у разі недостатньої перевірки або неправильної реалізації захисту.

Таблиця 2.1 - Типові вектори атак у межах архітектури системи

Етап системи	Можлива загроза	Опис ризику
Публікація firmware	Підміна файлу перед реєстрацією	Якщо зловмисник має доступ до публікації
Передача firmware	Перехоплення та підміна при доставці	Firmware не перевіряється належним чином
Верифікація firmware	Підміна хешу вручну	Якщо не звіряється з блокчейном
Завантаження застарілої версії	Rollback - атака	Якщо не перевіряється версія або статус
Сторонній запис у контракт	Несанконована публікація	Якщо немає перевірки ролі

Таким чином, сформована модель загроз дозволяє цілісно оцінити потенційні ризики на всіх етапах життєвого циклу прошивки - від моменту її створення до встановлення на IoT-пристрій. Визначені вектори атак та відповідні критерії безпеки створюють основу для подальшого проєктування механізмів захисту, що дозволяє забезпечити достовірність походження firmware, її незмінність, актуальність та захищеність від спроб підміни або повторного використання скомпрометованих версій. У наступному розділі ці критерії будуть покладені в основу архітектурних рішень та практичної реалізації блокчейн-орієнтованої системи автентифікації прошивок.

2.2.1 Актуальні вектори атак на етапі оновлення firmware

Оновлення прошивки - це процес модернізації прошивки, встановленої на пристрої, шляхом зміни її коду. Оновлення прошивки зазвичай надходять від виробника пристрою і призначені для виправлення помилок, підвищення продуктивності, посилення безпеки та впровадження нових функцій. Спосіб оновлення прошивки залежить від пристрою, а оновлення можуть надходити різними каналами, наприклад, бездротовим шляхом через Wi-Fi, USB-з'єднання або за допомогою комп'ютерного програмного забезпечення, наданого виробником [8]. Саме тому кожен компонент системи - від моменту створення firmware до її перевірки - має бути розглянутий з погляду потенційних ризиків.

Перша категорія загроз пов’язана з етапом публікації firmware. Якщо смарт-контракт дозволяє додавати записи без авторизації, зловмисник може опублікувати підроблений хеш і ввести в оману користувача. Ще гірший варіант - якщо запис можна змінити або видалити після публікації. Це дозволило б атакуючому "очистити" історію або підмінити офіційний запис на свій. Саме тому в системі передбачена авторизація лише для виробника, а самі дані записуються у блокчейн, де зміни постфактум неможливі [9].

Другий небезпечний момент - етап передачі firmware користувачу або IoT-пристрою. Якщо файл прошивки передається незахищеним каналом (наприклад, без підпису або безпосередньої перевірки), його легко перехопити і підмінити на шкідливий аналог. Це може статися під час завантаження з неперевіреного сайту або з фейкового оновлювального сервера. Навіть якщо хеш прошивки зберігається у контракті, користувач або пристрій повинні обов’язково звіряти файл із записом, а не покладатись на "надійне джерело"[10].

Окрему увагу варто приділити RollBack-атаці. У ній зловмисник примушує систему встановити стару версію firmware, в якій присутні відомі вразливості. Якщо система перевіряє лише хеш, але не враховує статус revoked

або номер версії - така атака цілком можлива. У спроєктованій системі для цього передбачено окреме поле revoked, що дозволяє адміністративно помічати застарілі або небезпечні прошивки як непридатні до використання. При перевірці цей статус обов’язково враховується.

2.2.2 Основні вимоги до безпеки системи

Критерії безпеки формуються на основі сучасних загроз, практик кіберзахисту та специфіки використання прошивок у середовищах з обмеженими ресурсами.

Насамперед, система повинна гарантувати цілісність firmware-файлів. Це означає, що будь-який файл, що перевіряється, має бути однозначно ідентифікований за допомогою криптографічного хешу. Обчислений хеш повинен повністю збігатися з тим, що зберігається у реєстрі смарт-контракту. У випадку найменшої модифікації - навіть на рівні одного байта - хеш не співпаде, і система повинна повідомити про порушення цілісності [11].

Другим критично важливим критерієм є автентичність джерела публікації прошивки. Система має надійно обмежити функцію додавання нових записів лише до довірених суб’єктів - зокрема, адміністратора або виробника пристрою. Це реалізується через механізм контролю доступу у смарт-контракті: тільки транзакції, що надійшли з попередньо дозволених адрес, можуть викликати функції реєстрації або відкликання firmware. Таким чином, гарантується, що записи в реєстрі можуть бути створені лише перевіреними суб’єктами [12].

Особливе значення має незмінність даних, що вже опубліковані. Система має базуватися на принципі, згідно з яким жодного разу зареєстровану інформацію не можна змінити або видалити. Цей критерій реалізується завдяки використанню блокчейн-платформи, що забезпечує незворотність транзакцій і

збереження повної історії дій у мережі. У такий спосіб виключається можливість фальсифікації або прихованої зміни відомостей про firmware.

У системі також буде реалізований механізм відкликання прошивки. У випадку виявлення вразливостей або релізу нової, безпечнішої версії, адміністратор повинен мати можливість позначити стару версію як недійсну. Для цього у структурі запису передбачено спеціальне поле статусу, яке сигналізує клієнтському застосунку про те, що прошивку більше не рекомендовано використовувати.

Суттєвим аспектом є забезпечення прозорості процесу перевірки. Кожен користувач або пристрій має мати можливість незалежно звернутися до смарт-контракту, зчитати публічну інформацію про прошивку і самостійно переконатись у її дійсності. Для цього функції перегляду в контракті мають бути позначені як відкриті, без потреби підпису чи витрат газу. Це дозволяє будь-кому - без додаткових дозволів - здійснити перевірку firmware на основі блокчейн-реєстру [13]. Також важливою є автоматизованість усіх перевірок. Користувач не повинен вручну шукати хеш чи зіставляти значення - за нього це виконує програмний модуль, що обчислює хеш, звертається до контракту, аналізує відповідь, перевіряє статус і, за потреби, підтверджує підпис. Автоматизація процесу підвищує точність перевірки та спрощує її застосування у реальному середовищі.

Таким чином, розглянуті критерії забезпечують комплексний підхід до захисту firmware і закладають архітектурну основу системи, де довіра базується не на централізованому сервері, а на відкритому незмінному реєстрі, підкріпленому криптографією. Дотримання цих принципів дозволяє досягти високого рівня безпеки при оновленні IoT-пристроїв і зменшити ризик їх компрометації внаслідок атаки через firmware.

2.3 Архітектура системи

Система використовуватиме децентралізовану модель перевірки цілісності даних на основі технології блокчейн. Такий підхід має низку принципових переваг над традиційними централізованими схемами контролю. У централізованій системі вся довіра зосереджується в одній точці, тому його компрометація ставить під загрозу функціонування всієї моделі безпеки.

Централізовані сервери для оновлення прошивок є привабливою мішенню для атак. Злам такого сервера дозволяє зловмисникам поширювати підроблені оновлення або спотворені дані в усю інфраструктуру. Блокчейн усуває цю уязвимість, розподіляючи довіру між багатьма вузлами мережі. Кожен запис, доданий до блокчейну, розповсюджується та дублюється на множині вузлів, через що несанкціоновано змінити чи видалити інформацію практично неможливо. Завдяки прозорості, незмінності та відсутності одноосібного контролера, блокчейн забезпечує підвищений рівень захисту даних і стійкість системи перевірки прошивок [14].

Однією з ключових властивостей блокчейн-реєстру є його незмінність. Інформація, одного разу додана до ланцюга блоків, не може бути потім потай змінена чи вилучена. Це означає, що всі дані про перевірки цілісності та оновлення прошивок зберігаються в надійному місці і доступні для незалежного аудиту у будь-який момент часу. Блокчейн веде детальний хронологічний журнал кожної операції, оновлення та перевірки, завдяки чому виробники й користувачі можуть простежити історію змін, перевірити автентичність прошивок та переконатися у дотриманні вимог безпеки. Можливості блокчейну також гарантує, що записи не можуть бути підроблені заднім числом, забезпечуючи невідворотність подій. Таким чином, найменша спроба фальсифікації даних або несанкціонованого втручання в ПЗ пристрою буде виявлена через невідповідність криптографічних хешів, а факт порушення цілісності залишиться назавжди зафіксованим у спільному реєстрі [15].

Для реалізації всіх зазначених властивостей архітектура системи передбачає чітку взаємодію між IoT-пристроєм та виробником, - кожен з яких виконує свою роль у забезпеченні цілісності прошивок без потреби в централізованому довіреному посереднику.

IoT-пристрій виступає кінцевим елементом системи та першим рівнем перевірки цілісності. Він повинен мати змогу обчислювати хеш своєї прошивки та звертатися до блокчейну безпосередньо або через шлюз, щоб перевірити її справжність за збереженими там даними [16]. Під час роботи або перед встановленням оновлення пристрій обчислює хеш поточної, або нової прошивки і звіряє його з еталонним значенням, отриманим із блокчейну. Лише у випадку повного збігу хешів оновлення вважається автентичним і дозволяється до інсталяції на пристрої. Якщо ж виявлено невідповідність тобто потенційне порушення цілісності, пристрій блокує встановлення підозрілого firmware та дає сигнал тривоги про можливу компрометацію програмного забезпечення. Таким чином, IoT-пристрій самостійно контролює цілісність власного програмного забезпечення, спираючись на довірені еталонні дані з блокчейну, і негайно реагує на будь-які порушення.

Виробник готує дані для перевірки цілісності ще до випуску оновлення. На етапі складання прошивки в контрольованому середовищі обчислюється її криптографічний хеш. Цей хеш потім підписується закритим ключем виробника і зберігається в смарт-контракті через спеціальну транзакцію. Таким чином, у блокчейні фіксується хеш, за яким надалі можна буде перевіряти справжність і незмінність прошивки під час її оновлення або використання. Після включення до блокчейну цей еталонний відбиток стає незмінним референтом для всіх учасників системи, тобто жоден зловмисник не зможе непомітно підмінити його іншим значенням завдяки криптографічному захисту та консенсусним правилам реєстру. Отже, виробник забезпечує початкову довіру, заносячи достовірні дані про кожну перевірену версію прошивки до блокчейну; надалі будь-який пристрій або користувач може звірити фактичну

версію ПЗ із цією записаною еталонною інформацією і впевнитися в її автентичності.

Користувач пристрою завдяки блокчейн-орієнтованій системі отримує можливість незалежно пересвідчитися, що на пристрої встановлено автентичне, не модифіковане сторонніми особами програмне забезпечення. Інформація, збережена у публічному реєстрі блокчейну, є доступною для перевірки уповноваженими особами без необхідності довіряти внутрішнім звітам або централізованим серверам. Зокрема, користувач через веб-інтерфейс може отримати від пристрою обчислене ним поточне хеш-значення прошивки і порівняти його з еталонним хешем, опублікованим виробником у смарт-контракті. Якщо система виявляє невідповідність, користувач негайно одержує сповіщення про проблему та може вжити необхідних заходів

Смарт-контракт, котрий розгорнутий у блокчейн-мережі, виконує функції автономного контролера та сховища даних для всієї системи. Він містить у собі логіку перевірки цілісності, як і коли верифікуються хеші прошивок, хто має право додавати нові дані, які дії слід виконувати у разі виявлення невідповідностей. У смарт-контракті зберігаються хеш версії прошивки, що їх публікують виробники, а також, за потреби, інша інформація: список авторизованих пристроїв, результати перевірок, логи подій. Смарт-контракт приймає транзакції від пристроїв та виробників і автоматично виконує закладені перевірки. Наприклад, коли пристрій надсилає власний обчислений хеш для верифікації, контракт порівнює його з еталонним значенням відповідної версії. У разі збігу контракт відмічає перевірку як успішну; якщо ж виявлено розбіжність, він фіксує цей факт у реєстрі та генерує подію тривоги, яка може бути відстежена системою моніторингу. Оскільки смарт-контракт працює децентралізовано (його копії виконуються всіма вузлами блокчейну), жоден зловмисник не здатен непомітно змінити логіку перевірки або підробити її результати. Це гарантує неупередженість контролю, всі учасники системи довіряють результатам, перевірки системи.

Отже, блокчейн-орієнтована архітектура перевірки цілісності firmware IoT-пристроїв забезпечує високий рівень довіри та безпеки за рахунок розподіленої моделі збереження даних і автоматизації контролю. Усі учасники системи – від виробника до кінцевого користувача – покладаються на спільний незмінний реєстр, а не на закриті централізовані механізми, що усуває багато вразливостей традиційних підходів, також архітектура, побудована за описаними принципами, є гнучкою щодо середовища впровадження, вона може бути адаптована під різні умови експлуатації та масштаби мереж.

2.4 Алгоритм перевірки цілісності та автентичності прошивок

Перевірка автентичності та цілісності firmware є необхідною умовою безпечної роботи будь-якої вбудованої або IoT-системи. Основою такої перевірки виступають криптографічні механізми хешування, цифрового підпису та незмінний реєстр прошивок, реалізований у смарт-контракті блокчейну. Хеш-функція забезпечує можливість виявлення будь-яких змін у файлі, оскільки навіть мінімальна модифікація призводить до формування нового хеш-значення. Цифровий підпис підтверджує походження прошивки, а реєстр у блокчейні фіксує затверджені версії та їхній статус.

Смарт-контракт виступає джерелом достовірної інформації про прошивки. Під час реєстрації виробник або розробник публікує хеш, метадані та статус нової версії. Запис зберігається у незмінному вигляді, що виключає можливість несанкціонованого редагування та дозволяє перевіряти справжність прошивки у будь-який момент [17]. Перевірка може виконуватися як на стороні пристрою, так і за допомогою користувацького клієнта. У типових умовах пристрій, отримавши файл прошивки, обчислює його хеш (наприклад, SHA-256), після чого звертається до смарт-контракту для пошуку відповідного запису. Якщо хеш присутній у реєстрі та має статус «активна», прошивка

вважається достовірною. У разі невідповідності хешів або якщо статус позначено як «відкликана» чи «застаріла», встановлення блокується.

Користувач також може виконати перевірку вручну за допомогою програмного клієнта. Клієнтська програма обчислює хеш отриманого файла, після чого надсилає запит до смарт-контракту. На основі відповіді визначається, чи є ця версія офіційною, дозволеною до використання та не відкликаною виробником. Якщо статус підтверджує справжність прошивки, користувач отримує повідомлення про її валідність. У протилежному разі файл вважається потенційно небезпечним.

Таким чином, алгоритм перевірки поєднує в собі хешування, підтвердження цифрового підпису та звернення до децентралізованого реєстру. Кожен з етапів дозволяє виявити модифікації файлу або неавторизовані версії firmware. Якщо будь-яка перевірка не проходить, система негайно припиняє процес встановлення, що забезпечує високий рівень захисту.

Таблиця 2.2 - Сумісність IoT-пристроїв із блокчейн-верифікацією прошивки

Тип пристрою	Обчислювальні ресурси	Можливість мережевої взаємодії	Підтримка хешування та API	Доцільність використання
1	2	3	4	5
Примітивний сенсор	Низькі	Нерегулярна або через шлюз	Ні	Ні
Розумна розетка	Низькі	Стабільне Wi-FI з`єднання	Так	Так
Камера відеоспостере-ження	Середні	Повноцінна підтримкка мережі	Так	Так

Кінець таблиці 2.2

1	2	3	4	5
Контролер на базі Linux	Високі	Повноцінна підтримка мережі	Так	Так
Сенсор LoRa/LPWAN	Низькі	Обмежена передача даних	Немає	Ні

Аналізуючи таблицю, можна зробити висновок, що розроблена система перевірки прошивки може застосовуватися для широкого класу IoT- та вбудованих пристроїв. До них належать мікроконтролерні модулі, сенсорні вузли, виконавчі блоки, промислові контролери та інші пристрої, які потребують регулярного оновлення firmware і здатні виконувати базові криптографічні операції, такі як обчислення хешу чи перевірка цифрового підпису.

Для демонстрації роботи системи у даній роботі використовується узагальнена прошивка для вбудованої системи, що дозволяє відтворити типовий цикл перевірки цілісності та автентичності. Будуть проводитися такі дії як формування хешу, взаємодія зі смарт-контрактом і підтвердження статусу прошивки. Такий підхід забезпечує універсальність моделі та показує, що запропонований механізм може бути застосований до будь-якого пристрою, який підтримує встановлення та оновлення firmware.

2.5 Проєктування структури смарт-контракту

Оновлення прошивок у традиційних системах здійснюються централізовано через сервер виробника, що створює ризик виникнення єдиної

точки відмови та ускладнює гарантування цілісності firmware. Щоб уникнути цих недоліків, у розробленій системі використано блокчейн Ethereum та смарт-контракт, написаний мовою Solidity. Смарт-контракт виконує роль децентралізованого реєстру метаданих прошивок IoT-пристроїв, забезпечуючи їхню перевірку на справжність. Усі дані про версії прошивок зберігаються у блокчейні, що виключає несанкціоновану зміну записів, підвищує прозорість оновлень та створює довіру до процесу.

Структура смарт-контракту формує запис прошивки у вигляді набору полів, що визначають її властивості та статус:

- криптографічний хеш (SHA-256) - унікальний ідентифікатор прошивки; будь-яка зміна байтів призводить до іншого значення;
- цифровий підпис виробника, сформований від хешу, – підтверджує справжність походження прошивки;
- час публікації, зафіксований через номер блоку Ethereum, який визначає момент реєстрації;
- Ethereum-адреса публікатора, що дозволяє перевірити, хто саме додав запис;
- статус (revoked / active), який відображає чинність версії та дає змогу оперативно позначати виявлені або скомпрометовані прошивки як заборонені.

Таким чином, структура забезпечує можливість створення безпечного та прозорого механізму оновлення firmware. IoT-пристрої отримують можливість самостійно перевіряти справжність нових прошивок без необхідності довіряти централізованим серверам, покладаючись лише на криптографію та консенсус розподіленої мережі. Такий підхід відповідає сучасним тенденціям підвищення кіберстійкості IoT-інфраструктури й актуальний для систем, де компрометація firmware може мати критичні наслідки (медицина, промисловість, транспорт тощо).

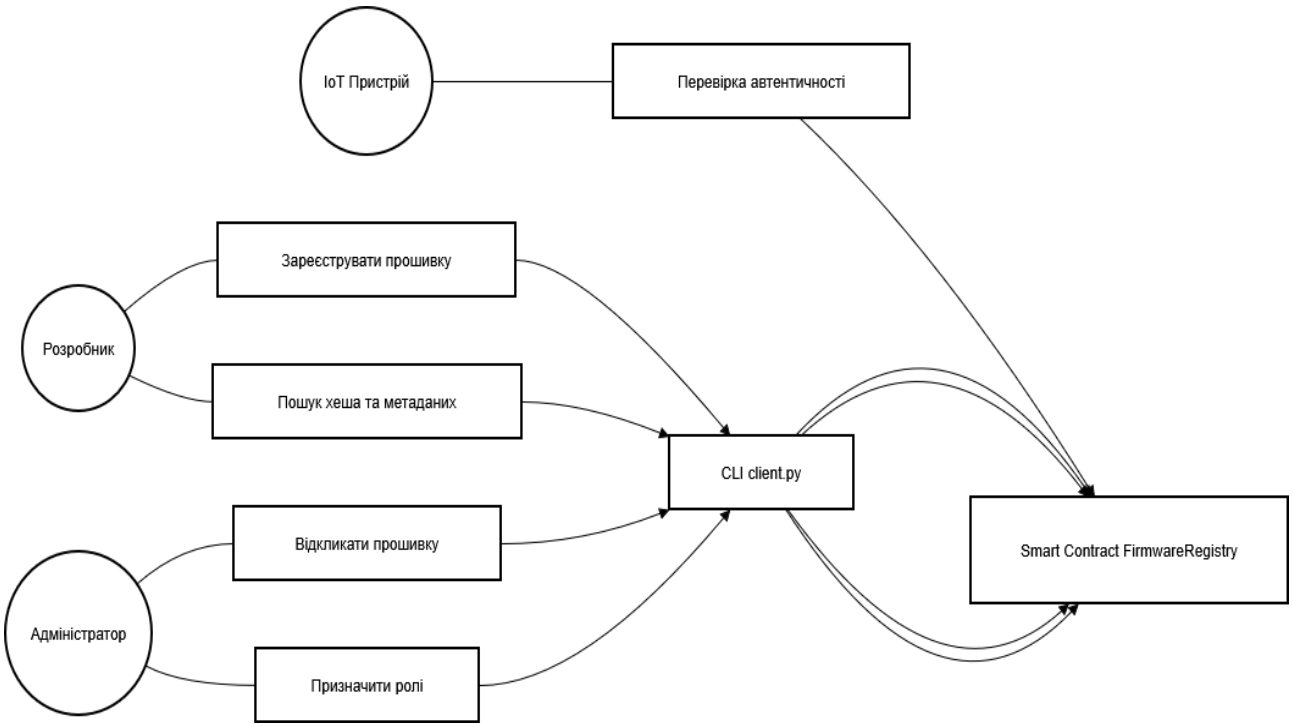


Рисунок 2.3 - Діаграма варіантів використання системи перевірки прошивок

На діаграмі показано основні дії, які виконують учасники системи. Розробник реєструє нові прошивки та здійснює пошук їхніх метаданих у смарт-контракті. Адміністратор керує ролями та може відкликати вразливі версії. IoT-пристрій використовує функцію перевірки автентичності для підтвердження, що прошивка відповідає еталонним даним у блокчейні. Клієнтська програма забезпечує взаємодію зі смарт-контрактом, передаючи запити та отримуючи відповіді.

2.6 Обґрунтування вибору технологій та інструментів

Для проєктування системи перевірки прошивки IoT-пристроїв обрано Ethereum-платформу і відповідний стек розробки. Для локального тестування смарт-контракту використовується Ganache він є персональним блокчейн

Ethereum, який запускається одним кліком та емітує роботу повноцінного Ethereum-ланцюга. Ganache створює тестову мережу з фіктивними обліковими записами та балансам, що дозволяє перевіряти контракти без реальних транзакцій і витрат на газ. Це значно економить час і кошти розробки, розгортання контрактів відбувається миттєво, а затримки очікування блокчейну відсутні. Крім того, ізольована приватна мережа Ganache забезпечує захист від випадкового підключення до публічного Ethereum та дозволяє повністю контролювати середовище тестування (налаштування часу блоків, логування транзакцій тощо).

Для взаємодії з блокчейном вибрано бібліотеку Web3.py. Він надає методи для відправлення транзакцій, виклику функцій контрактів і читання даних з блокчейну без необхідності використання браузерного гаманця MetaMask. Це дозволяє реалізувати програмний зв’язок між Python-додатком та мережею Ethereum через JSON-RPC (Remote Procedure Call). Бібліотека Web3.py має простий і зрозумілий API, що спрощує розробку децентралізованих застосунків. Використання Web3.py дає змогу програмно опрацьовувати транзакції (підписувати їх приватним ключем) й ефективно керувати контрактом із сервера чи десктопа – що особливо зручно для CLI-інструментів або автономних додатків.

Міжпланетна файлова система (IPFS) являє собою зміну парадигми в тому, як ми концептуалізуємо та реалізуємо файлові системи. IPFS пропонує децентралізований підхід до зберігання, адресації та розповсюдження контенту через однорангову мережу. На відміну від традиційних клієнт-серверних моделей, які домінують в сучасному Інтернеті, IPFS передбачає мережу, де доступ до контенту здійснюється на основі того, що він собою являє, а не де він знаходиться [18]. Це забезпечує незмінність даних, вміст можна витягти за хешем, а отриманий файл можна перевірити на відповідність запросу шляхом повторного обчислення CID. Застосування IPFS відповідає вимогам безпеки, жодна модифікація файлу неможлива без зміни його хеша, що гарантує цілісність оновлень прошивки.

Основні переваги обраних інструментів та їхня відповідність вимогам безпеки і спрощеній локальній реалізації:

- Ganache дозволяє ізолювати приватний блокчейн, ліквідує витрати на газ та очікування при розгортанні контрактів. Розробка і відлагодження відбуваються повністю локально, без ризику випадкових публічних транзакцій;

- Python у цій роботі служить основним інструментом для реалізації клієнтської частини системи. Використання Web3.py забезпечує зручний доступ до смарт-контракту через JSON-RPC та дозволяє виконувати ключові операції, пов'язані з обробкою прошивки: обчислення хешу, формування запиту до блокчейну, отримання даних про статус firmware та надсилання транзакцій для її реєстрації або відкликання. Мова Python також надає готові криптографічні засоби, включно з реалізацією SHA-256, що робить її придатною для створення компактного CLI-клієнта без використання додаткових інструментів чи графічних фреймворків. Уся взаємодія з системою відбувається через окремий Python-скрипт, який забезпечує повний цикл роботи користувача з прошивкою та смарт-контрактом;

- надійна взаємодія з блокчейном, Web3.py є усталеною бібліотекою для Ethereum з простим API. Вона полегшує програмне надсилання транзакцій та читання стану смарт-контракту, що спрощує інтеграцію блокчейну в бекенд-сервіс;

- контроль цілісності забезпечує IPFS, контент-адресацію файлів, тобто будь-які зміни у прошивці змінять її хеш. Це відповідає принципам безпеки оновлень: якщо файл прошивки змінено, його CID зміниться, і система виявить невідповідність. Таким чином, навіть демонстраційне використання IPFS (наприклад, зберігання лише хешів на блокчейні);

- спрощена локальна реалізація**,** всі обрані компоненти можуть бути встановлені й запущені локально. Ganache і Python-средовище не вимагають розгортання у хмарі, а Web3.py працює через локальний RPC. Це дозволяє у ході дипломного проєкту швидко створювати прототип без залежності від зовнішніх мереж чи сертифікатів. Одночасно приватна локальна мережа

блокчейну відповідає принципам безпеки – її вузли відомі та контролюються розробником, що унеможливлює сторонні атаки у демонстраційному середовищі.

Для моделювання поведінки системи та переходів між різними статусами прошивки доцільно використати діаграму станів, яка відображає життєвий цикл firmware у смарт-контракті.

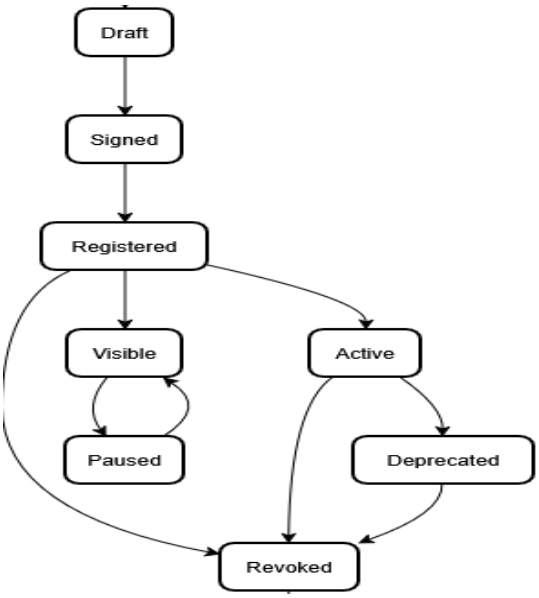


Рисунок 2.6 - Діаграма станів прошивки у смарт-контракті

На діаграмі представлено основні стани, через які проходить запис про прошивку у смарт-контракті: від початкового стану «Draft» до «Signed» та Після реєстрації прошивка може перейти у стан «Active» або бути відкликана Така модель відображає логіку роботи контракту та забезпечує контроль цілісності firmware упродовж повного життєвого циклу.

Таким чином, обраний стек технологій та модель станів смарт-контракту забезпечують надійну, прозору та відтворювану перевірку прошивок IoT-пристроїв. Система здатна виявляти підроблені або скомпрометовані версії, контролювати їхній статус та гарантувати цілісність програмного забезпечення на всіх етапах його життєвого циклу, що є ключовою вимогою до безпечних IoT-рішень.

2.7 Сценарії застосування системи перевірки прошивок у різних середовищах

Розроблена система може застосовуватися у широкому спектрі IoT-та вбудованих рішень, які потребують надійного механізму контролю цілісності та автентичності програмного забезпечення. У сучасних розподілених інфраструктурах такі пристрої використовуються як у побутових умовах, так і у промислових або критичних середовищах, де навіть незначна модифікація прошивки може призвести до порушення працездатності системи або створення додаткових ризиків безпеці.

Застосування запропонованої моделі доцільне для пристроїв, що працюють на базі мікроконтролерів та регулярно отримують оновлення firmware. До таких рішень належать сенсорні модулі, виконавчі вузли автоматизації, мережеві шлюзи, контролери невеликих локальних систем, а також пристрої домашньої автоматизації. Незалежно від конкретного типу обладнання, усі вони мають спільну потребу у гарантії того, що встановлюване оновлення не було змінено, підмінено або пошкоджено під час передачі.

У локальних IoT-мережах система може виконувати роль джерела довіри під час встановлення або оновлення прошивки. У таких умовах блокчейн використовується як реєстр еталонних хешів та статусів firmware, що дозволяє пристроям у мережі перевіряти достовірність файла перед інсталяцією. Подібний підхід забезпечує додатковий рівень контролю у домашніх, офісних або лабораторних середовищах, де декілька пристроїв можуть взаємодіяти через спільний контролер або локальний сервер оновлень [19].

У промислових системах та середовищах IoT перевірка цілісності firmware має ще важливіше значення. Робота сенсорів, контролерів приводу, вузлів телеметрії та інших компонентів часто пов’язана з виробничими процесами, де порушення алгоритмів або встановлення модифікованого програмного забезпечення може призвести до небажаних наслідків. У таких

умовах система перевірки дозволяє підвищити довіру до джерела оновлень та мінімізувати ризики поширення скомпрометованої прошивки.

Таким чином, модель, що була розроблена у даній роботі, не прив’язана до конкретного типу IoT-пристрою. Її можна адаптувати до будь-якої вбудованої системи, яка використовує firmware та потребує надійного механізму перевірки її походження та незмінності. Універсальність запропонованого рішення робить його придатним як для навчальних прототипів, так і для реальних інфраструктурних рішень.

3 РЕАЛІЗАЦІЯ БЛОКЧЕЙН АВТЕНТИФІКАЦІЇ

3.1 Експериментальне середовище та початкові налаштування

Для лабораторної демонстрації використано локальний блокчейн-ланцюг Ganache, який емулює середовище Ethereum із набором тестових акаунтів, транзакцій і початкових балансів. Це середовище забезпечує можливість повної емуляції розгортання смарт-контракту, виконання операцій із хешами прошивок і фіксації транзакцій без потреби у зовнішній мережі.

Інтерфейс Ganache надає візуальний контроль за блоками, хешами транзакцій і станом гаманців. На рис. 3.1 наведено зовнішній вигляд середовища Ganache, що демонструє структуру мережі, ідентифікатори акаунтів і поточні залишки на них.

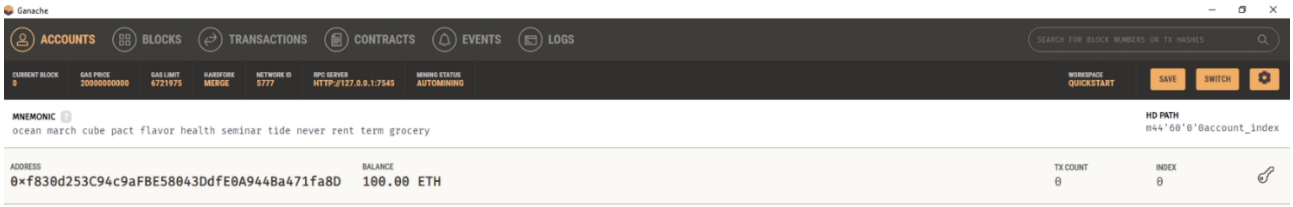


Рисунок 3.1 – Вигляд середовища Ganache

Ganache дає змогу швидко ініціалізувати приватний блокчейн-ланцюг із довільною кількістю акаунтів і миттєвим підтвердженням транзакцій, що робить його ідеальним інструментом для тестування смарт-контрактів. Саме у цьому середовищі було проведено початкове розгортання реєстру прошивок, визначення адміністратора та перевірку транзакцій на валідність.

Адміністративний суб’єкт визначається як власник пари ключів ECDSA, створених Ganache під час ініціалізації мережі. На рис. 3.2 наведено адресу адміністратора та відповідний приватний ключ, що застосовується виключно в демонстраційному середовищі.

Рисунок 3.2 – Адреса та приватний ключ демонстраційного адміністратору

Ключова пара адміністратора виконує роль кореня довіри у системі. Публічна адреса визначає власника реєстру, а приватний ключ використовується для формування цифрових підписів при реєстрації нових прошивок і виконанні транзакцій управління. Без наявності цього ключа реєстрація чи відкликання прошивки стає неможливою, що забезпечує однозначну автентичність дій адміністратора.

Конфігураційні параметри системи централізовано зберігаються у файлі .env. Цей файл містить змінні середовища, необхідні для з'єднання з вузлом блокчейну, а також для підписування транзакцій. На рис. 3.3 наведено приклад вмісту файлу .env, який включає основні параметри:

- RPC_URL - адреса RPC-ендпойнта Ganache (локальний вузол);
- ADMIN_PRIVATE_KEY - приватний ключ адміністратора, що застосовується лише для підпису;
- CHAIN_ID - ідентифікатор мережі, який забезпечує захист від повторного відтворення транзакцій у сторонніх ланцюгах.

Зберігання параметрів у .env дає змогу легко адаптувати систему до будь-якого середовища - від локальної симуляції у Ganache до тестових мереж чи реального Ethereum-ланцюга. Розробник може змінювати налаштування без

модифікації коду, що значно підвищує портативність і відтворюваність. Більше того, чутливі значення ніколи не розміщуються у відкритому доступі, адже .env зазвичай включається до .gitignore, що гарантує відсутність витоків під час командної розробки або інтеграції зі сторонніми сервісами.

Рисунок 3.3 – Наповнення файлу .env

Такі принципи конфігурації забезпечують високий рівень безпеки та спрощують процес розгортання системи у різних середовищах. Це також дозволяє гарантувати, що інструменти, які взаємодіють із блокчейном, зчитують усі необхідні параметри динамічно та завжди працюють у релевантному контексті.

Приватний ключ, зазначений у .env, передається до розгортального сценарію для ініціалізації підписувача. На рис. 3.4 наведено фрагмент коду файлу deploy_with_ethers.ts, де здійснюється створення об’єкта Wallet із приватного ключа, що дає можливість підписувати транзакції під час публікації контракту в мережу.

```
(async () => {
  const CONTRACT_NAME = "FirmwareRegistry";

  // ---- Налаштування для RPC fallback (Ganache) ----
  const RPC_URL = "http://127.0.0.1:7545";   // ваш Ganache RPC
  const PRIVATE_KEY = "0xbba4c07c2e370d429917c1d655321f2404e736a091c2b7494da2082cc7ccd610";

// 1) compile
  try { await remix.call("solidity", "compile"); } catch {}
```

Рисунок 3.4 – Використання приватного ключу в коді розгортання

У процесі розгортання смарт-контракту приватний ключ адміністратора інтегрується у виконуваний скрипт через безпечне читання параметрів із файлу конфігурації .env. Такий підхід дозволяє ізолювати секретні дані від вихідного коду та мінімізувати ризики їх витоку під час розробки. Механізм ініціалізації підписувача забезпечує можливість автономного створення транзакцій і формування валідних ончейн-записів без участі зовнішніх інтерфейсів. Завдяки цьому процес розгортання стає відтворюваним, контрольованим і придатним для автоматизації в межах DevOps-процедур.

3.2 Розгортання та фіксація артефактів контракту

Після налаштування середовища виконано безпосереднє розгортання смарт-контракту FirmwareRegistry, який реалізує логіку збереження хешів і метаданих прошивок. Розгортання здійснюється за допомогою скрипта scripts/deploy_with_ethers.ts, що компілює код Solidity 0.8.19 із використанням бібліотеки OpenZeppelin ECDSA v4.9.0.

У процесі публікації створюється екземпляр контракту, який отримує власну унікальну адресу у ланцюзі. На рис. 3.5 представлено консольний вивід результату розгортання, де наведено адресу контракту, хеш транзакції та обсяг спожитого газу.

```
running scripts/deploy_with_ethers.ts ...
signer: 0xf830d253C94c9aFBE58043DdfE0A944Ba471fa8D
network: 1337
tx sent: 0x43882796c7d5bb2732b3ae0177419eb0f1a81e36412afaab33ddb698953527f2
FirmwareRegistry deployed at: 0x5DB92A6f35b856b8CcCD0678F7ee8b1eC55C12f1
receipt: 0x43882796c7d5bb2732b3ae0177419eb0f1a81e36412afaab33ddb698953527f2
gasUsed: 1882246
```

Рисунок 3.5 – Результат роботи розгортання смарт контракту

Журнал розгортання підтверджує успішне виконання всіх кроків - компіляцію, підпис транзакції та запис у блокчейн. Таким чином, створено базову основу для подальших взаємодій із реєстром прошивок та виконання адміністративних дій. Після розгортання отримана адреса контракту зберігається у текстовому файлі contract.address, що використовується клієнтськими застосунками для підключення до розгорнутого екземпляра (рис. 3.6). Це дає можливість усім офчейн-компонентам тобто Python-скриптам перевірки та публікації взаємодіяти з одним і тим самим реєстром без потреби повторного деплою.

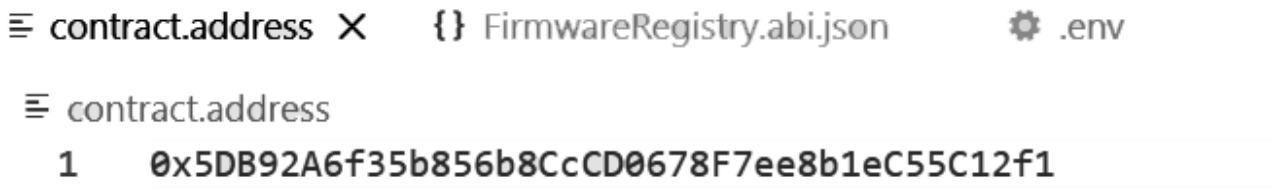


Рисунок 3.6 - Встановлення адреси контракту

Збереження адреси у файлі є необхідним етапом для забезпечення детермінованої роботи всієї системи. Будь-який користувач або пристрій, маючи цей файл і ABI-опис контракту, може виконати запит до блокчейну й отримати офіційні дані про прошивку, її хеш, час публікації та статус («official», «revoked» або «not found»). Це створює базу для подальших розділів, де демонструється реєстрація, перевірка та відкликання прошивок у реальному середовищі експлуатації.

Інтерфейс ABI (Application Binary Interface) є формальною специфікацією структури функцій, типів даних та подій, реалізованих у смарт-контракті. У контексті даної системи ABI зберігається у файлі FirmwareRegistry.abi.json, повна версія якого наведена у Додатку Г. Саме на основі цього опису офчейн-клієнти, такі як Python-скрипти client.py і verify_firmware.py, можуть

детерміновано формувати транзакції, викликати методи смарт-контракту, читати дані з реєстру та проводити перевірки.

Наявність актуальної ABI-структури разом із адресою контракту забезпечує можливість повторюваної та коректної взаємодії з уже розгорнутим екземпляром смарт-контракту. Без ABI офчейн-інструменти не здатні б були інтерпретувати формати вхідних та вихідних параметрів, що робить ABI критично важливим елементом усієї системи керування прошивками.

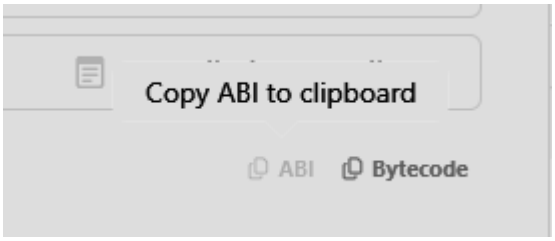


Рисунок 3.7 – Копіювання ABI структури після компіляції

Зображене на рис. 3.7 підтверджує, що ABI отримано безпосередньо з компілятора Solidity і збережено у файловій системі. Такий спосіб гарантує узгодженість версій програмного коду та опису контракту, що є ключовою вимогою забезпечення відтворюваності

Після завершення розгортання смарт-контракту та фіксації його артефактів усі офчейн-компоненти системи отримують можливість виконувати реальні операції реєстрації, перевірки та відкликання прошивок. Щоб продемонструвати фактичний порядок взаємодій між розробником, клієнтською програмою, RPC-провайдером та смарт-контрактом, подано діаграма послідовності, яка відображає повний цикл публікації нової прошивки у блокчейні.

Саме цей сценарій є базовим для подальшої роботи системи, оскільки визначає, яким чином формується й потрапляє до реєстру перша версія firmware, її хеш, цифровий підпис та пов’язані метадані.

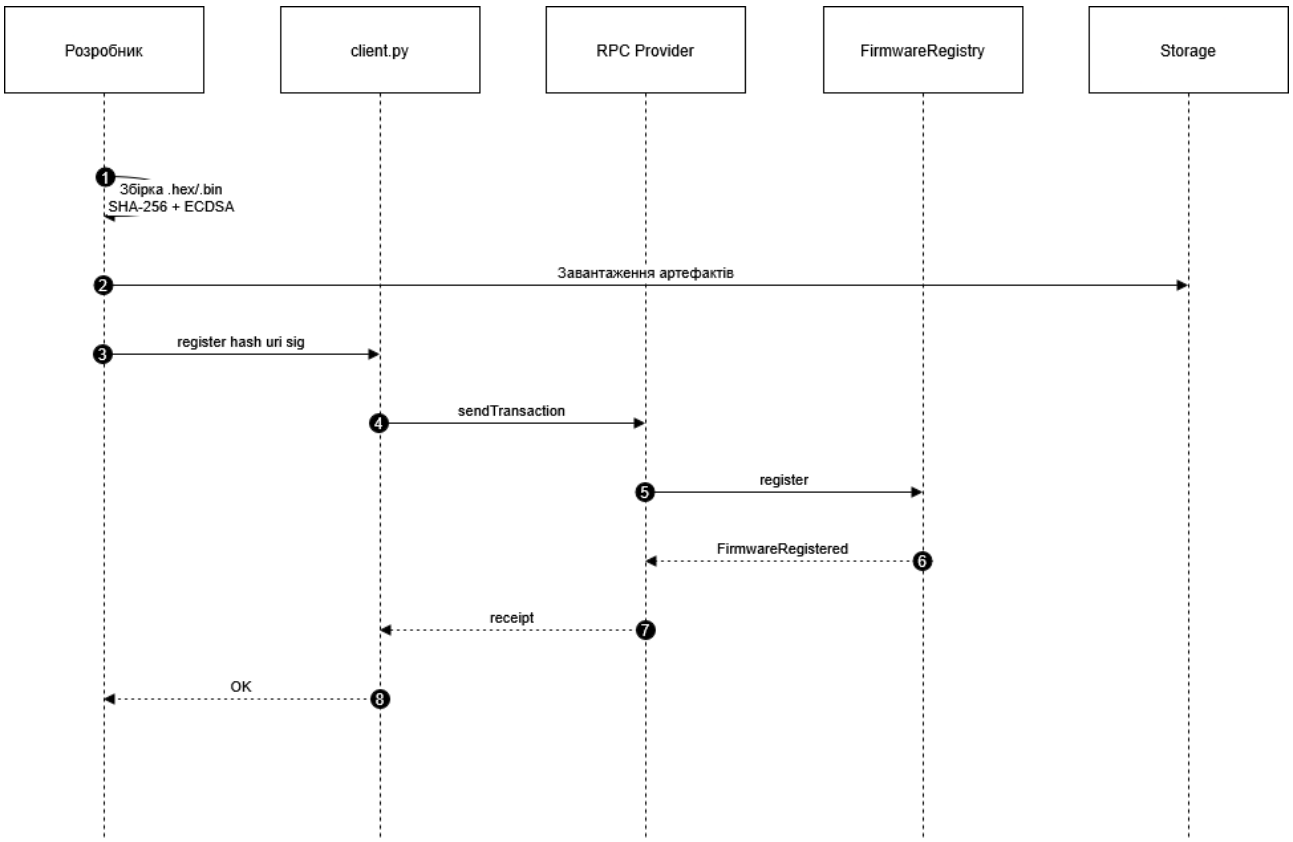


Рисунок 3.8 - Послідовність операцій під час реєстрації прошивки в смарт-контракті

На діаграмі показано, як розробник передає готову прошивку в клієнтський Python-скрипт, який обчислює її хеш та формує транзакцію register. Далі транзакція надсилається через RPC-провайдер до смарт-контракту FirmwareRegistry, який зберігає опублікований запис у блокчейні та генерує подію підтвердження. Після цього клієнтська програма отримує квитанцію виконання транзакції й передає розробнику результат реєстрації.

Таким чином, процес реєстрації прошивки повністю автоматизовано та забезпечено криптографічним захистом на всіх етапах. Наявність фіксованої послідовності дій гарантує відтворюваність операцій у тестовому та реальному середовищах, а успішна реєстрація firmware формує підґрунтя для подальших процедур перевірки та відкликання, які розглядаються у наступних підрозділах.

3.3. Модель артефактів прошивки та контент-адресація

Прошивка в системі розглядається як бінарний програмний артефакт, що є кінцевим результатом компіляції вихідного коду мікроконтролера, зокрема Arduino-скетчів. На рис. 3.9 наведено фрагмент вихідного коду прошивки, у якому міститься змінна FW_TAG. Цей тег використовується для індикативного позначення версії прошивки та дозволяє здійснювати базову ідентифікацію в процесі експлуатації.

```
22
23
24  void setup() {
25    Serial.begin(115200);
26    pinMode(CLK, OUTPUT);
27    pinMode(DIN, OUTPUT);
28    pinMode(CS, OUTPUT);
29
30    // Setup each MAX7219
```

Рисунок 3.9- Відрізок прошивки

Попри наявність текстового маркера версії, остаточним джерелом істини щодо цілісності файлу є виключно криптографічний хеш. Лише біт-у-біт відповідність бінарного вмісту гарантує захист від непомітних модифікацій, включно з додаванням шкідливих інструкцій чи зміною поведінки пристрою.

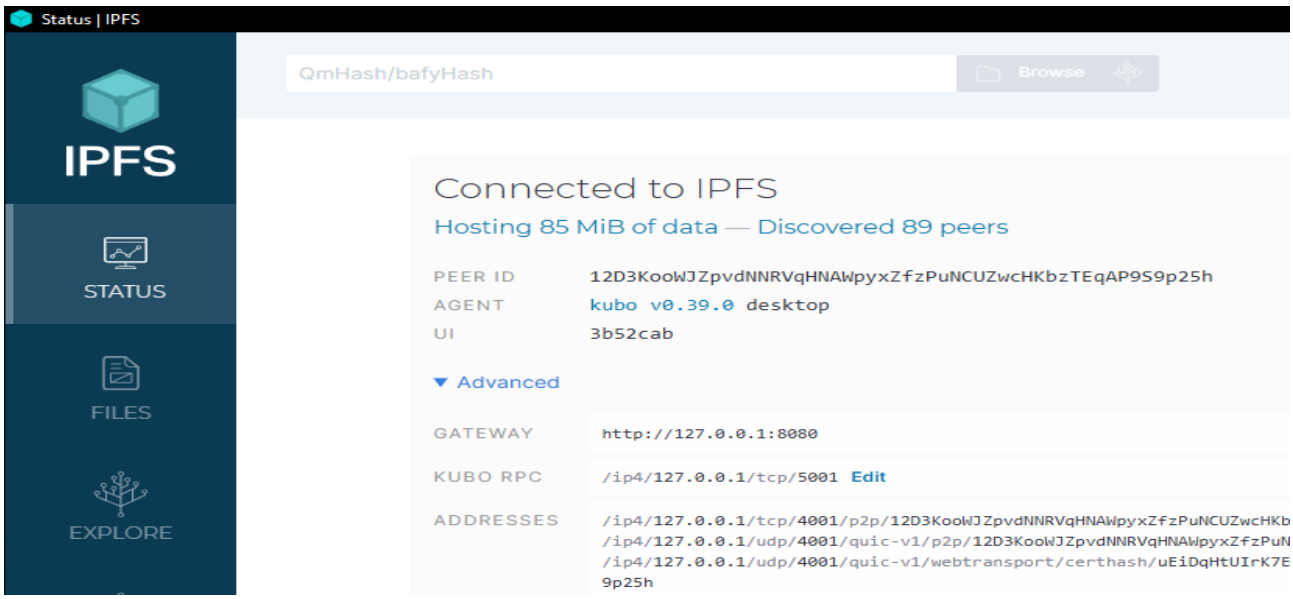


Рисунок 3.10 – Адреса IPFS

Бінарний файл прошивки після компіляції завантажується до децентралізованого сховища IPFS. На рис. 3.10 представлено URI виду ipfs://<CID>, що посилається на конкретну версію прошивки, а на рис. 3.11 показано результат фактичного завантаження файлу до IPFS Desktop.

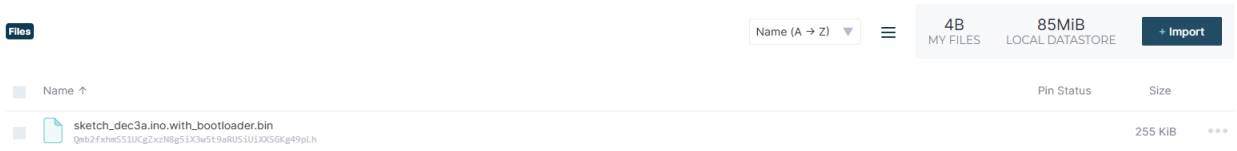


Рисунок 3.11 – Завантажений файл прошивки у IPSF Desktop

Механізм контент-адресації у IPFS гарантує, що будь-яка зміна хоча б одного байта у файлі призведе до генерації нового CID (Content Identifier). Таким чином, CID та SHA-256 є взаємопов'язаними ідентифікаторами, що відображають унікальний бітовий вміст. Власне, хеш, переданий у смарт-контракт, обчислюється таким чином:

$$H = SHA256(F) \in \{0,1\}^{\{256\}}$$

де F - бінарний вміст файлу прошивки. У Solidity цей хеш зберігається як bytes32, що дозволяє ефективно перевіряти відповідність хешу та відновлювати підпис. Наявність одночасно CID в IPFS і SHA-256 у блокчейні формує двофакторну модель контролю цілісності, у якій IPFS забезпечує незмінність самого вмісту, тоді як блокчейн гарантує незмінність запису, що описує цей вміст.

Поєднання контент-адресації IPFS та криптографічного хешування дозволяє створити надійну модель перевірки цілісності firmware під час її завантаження на пристрій. Щоб продемонструвати порядок виконання цієї перевірки в реальному сценарії OTA-оновлення, наведено діаграму послідовності, яка відображає обмін даними між IoT-пристроєм, сервером оновлень та смарт-контрактом.

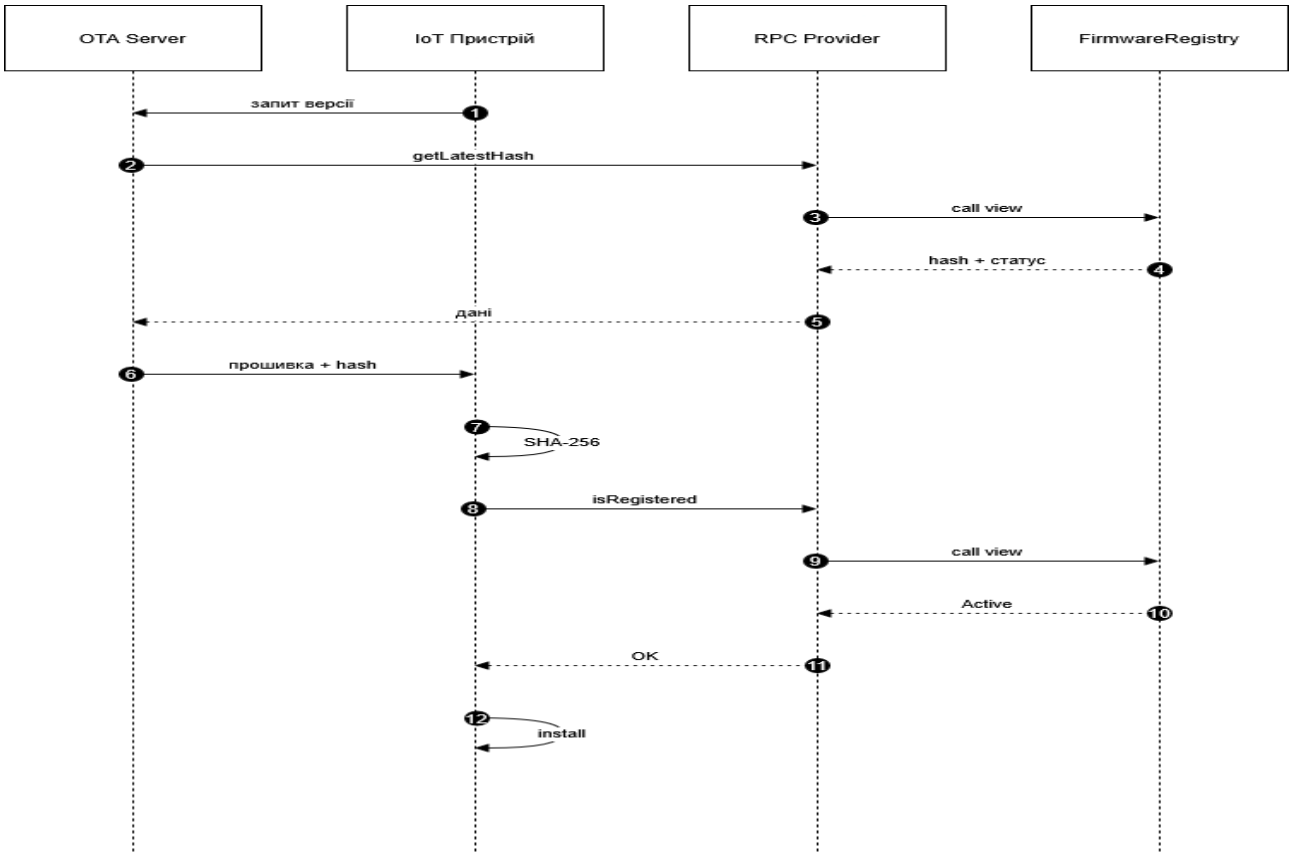


Рисунок 3.12 - Послідовність операцій під час OTA-перевірки прошивки

На діаграмі показано, як IoT-пристрій отримує метадані прошивки від OTA-сервера, обчислює її хеш та звертається до смарт-контракту для перевірки статусу та автентичності. Якщо хеш збігається з еталонним і статус прошивки є активним, пристрій продовжує встановлення оновлення. У разі невідповідності хешу або відкликаного статусу завантаження блокується, що унеможливлює встановлення модифікованого або небезпечного ПЗ.

Таким чином, модель контент-адресації IPFS у поєднанні з блокчейн-реєстром забезпечує двоетапний контроль цілісності firmware, а реалізований механізм OTA-перевірки гарантує, що на пристрій потрапляють лише офіційні й незмінені версії прошивок.

3.4 Смарт-контракт FirmwareRegistry та інваріанти безпеки

Структура запису в реєстрі прошивок охоплює всі необхідні атрибути для забезпечення цілісності, автентичності та відтворюваності даних про прошивку.

Контракт включає такі поля:

- hash: bytes32 - криптографічний SHA-256 хеш бінарного файлу;
- signature: bytes - ECDSA-підпис адміністратора над хешем у форматі EIP-191 personal_sign;
- ipfsHash: string - контент-адресаційний URI ipfs://<CID>;
- firmwareName та serialNumber - довільні ідентифікатори версії та серійної групи;
- uploadTime - момент публікації, фіксований блокчейном;
- publisher - адреса адміністратора, який здійснив реєстрацію;
- revoked: bool - ознака відкликаної прошивки.

Поведінка смарт-контракту ґрунтується на низці ключових інваріантів, що гарантують дотримання проєктних вимог до безпеки та цілісності даних. Насамперед визначається єдине джерело авторства, оскільки функція

registerFirmware містить модифікатор onlyAdmin, який унеможливлює додавання записів будь-ким, крім адміністратора.

Важливою складовою механізмів захисту є автентичність підпису, оскільки контракт самостійно відновлює адресу підписанта на основі повідомлення у форматі "\x19Ethereum Signed Message:\n32" || H" і приймає реєстрацію лише тоді, коли відновлена адреса збігається з адміністративною. Додаткову стійкість системи забезпечує принцип одноразовості хешу, адже повторна спроба зареєструвати той самий SHA-256 блокується, що унеможливлює перезапис або непомітну модифікацію вже існуючих записів. Окрім цього, контракт надає засоби керування життєвим циклом прошивки за допомогою методу revokeFirmware(H), який позначає відповідний запис як недійсний; зміна статусу миттєво відображається у відповідях getFirmwareStatus, забезпечуючи прозорість та контроль над усіма станами, у яких може перебувати артефакт прошивки. Після викладених положень доречно наголосити, що саме їх сукупність формує цілісну модель безпеки, у якій блокчейн відповідає за незмінність стану, механізми ECDSA гарантують автентичність авторства, а IPFS усуває ризики підміни або модифікації файлу поза межами ланцюга. У такій конфігурації смарт-контракт виступає фундаментом довіреної інфраструктури, що забезпечує передбачувані та формально гарантовані властивості системи.

Для демонстрації механізму відкликання прошивки та взаємодії між клієнтською програмою, смарт-контрактом і зовнішніми компонентами наведено діаграму послідовності, яка відображає процес зміни статусу прошивки на «revoked».

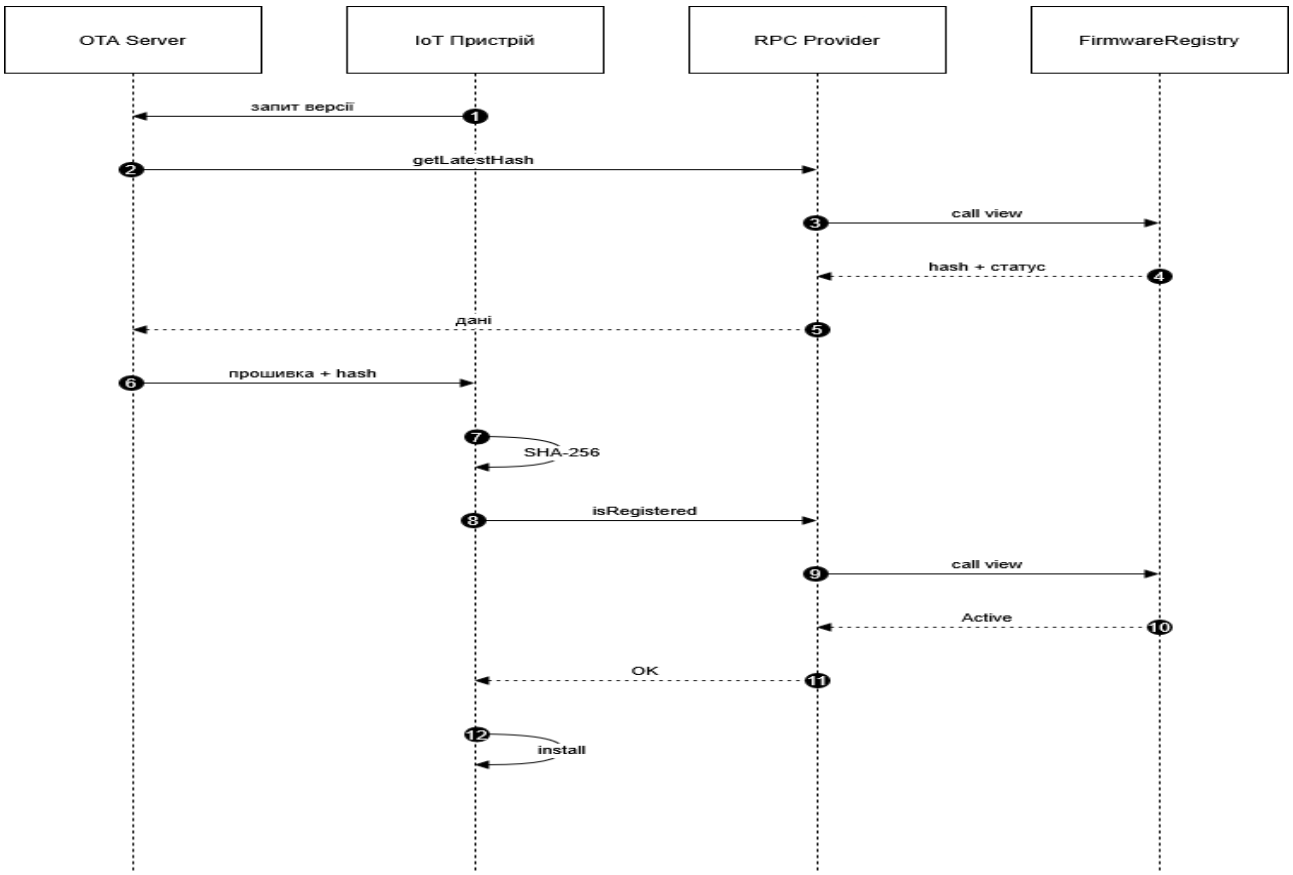


Рисунок 3.13 - Послідовність операцій під час відкликання прошивки в смарт-контракті

На діаграмі зображено процес відкликання прошивки, у якому адміністратор надсилає запит через клієнтську програму, що формує та підписує транзакцію revoke. Після передання через RPC-провайдер смарт-контракт змінює статус вибраної версії прошивки на «revoked», а подія відкликання реєструється у блокчейні. Це забезпечує негайне блокування використання застарілих або скомпрометованих версій як пристроями, так і зовнішніми інструментами перевірки.

Таким чином, інваріанти смарт-контракту та реалізовані функції зміни станів прошивки забезпечують цілісність і контроль життєвого циклу firmware. Відкликання версії відбувається прозоро, атомарно та під криптографічними гарантіями автентичності, що унеможливлює несанкціоноване втручання та формує надійну базу для подальшої експлуатації системи.

3.5 Процес реєстрації прошивки у системі

Процес реєстрації прошивки відбувається поза блокчейном, але з обов'язковим застосуванням криптографічних методів та підпису адміністратора.

Скрипт client.py register реалізує наступні кроки:

- обчислення хешу: H = SHA256(file);
- формування EIP-191-повідомлення на основі гекс-подання H;
- підписання повідомлення ключем ECDSA адміністратора;
- виклик registerFirmware(H, sig, name, serial, ipfs) з передачею хешу, підпису та URI з IPFS.

Успішна реєстрація формує незмінний запис у мережі, який може бути перевірено будь-якою стороною. У блокчейні зберігається не сама прошивка, а лише її хеш, підпис та метадані. Такий підхід забезпечує мінімальні витрати газу і водночас гарантує контроль походження та цілісності.

```
(venv) PS X:\programming\JS\ethereum\diploma> python client.py register sketch_dec3a.ino.with_bootloader.bin     --name "FW-
1.0"      --serial "UNO-001"     --ipfs "ipfs://Qmb2fxhmS51UCgZxzN8g5iX3w5t9aRUSiUiXXSGKg49pLh"
Registered. tx: aa124bf73d3ffbe0e515e8cac69422d34be0e87f32d7c7ad27c9048dbe3d6232
```

Рисунок 3.14 – Реєстрація прошивки у блокчейні

Публікація фіксується транзакцією, яка отримує власний хеш та включається до чергового блоку. Надалі всі компоненти системи можуть звертатися до реєстру для отримання статусу будь-якої прошивки.

Скрипт verify_firmware.py реалізує формальний детермінований алгоритм аудиту, який дозволяє будь-якому користувачеві або автоматизованому агенту підтвердити автентичність та цілісність прошивки на основі даних, збережених у блокчейні, а також на основі контент-адресації у системі IPFS. Алгоритм дозволяє перевіряти походження, незмінність і актуальність програмного

бінарника перед його встановленням на пристрій, що є необхідною умовою захисту IoT-екосистеми від підміни чи маніпуляцій.

Алгоритм перевірки працює з кількома вхідними параметрами, серед яких локальний файл прошивки F, адреса розгорнутого смарт-контракту та ABI-файл, що визначає його інтерфейс. За потреби може бути також активовано режим звірки контенту з IPFS, який підвищує рівень достовірності результатів. Процедура передбачає послідовне виконання кількох етапів. Спершу локально обчислюється криптографічний хеш $Hlocal=SHA256(F)$, який є унікальним відбитком вмісту файлу. Далі алгоритм звертається до реєстру, використовуючи функцію getFirmwareStatus(H_local), щоб отримати стан прошивки, який може набувати значень official, revoked або not found. Якщо запис існує, скрипт додатково зчитує метадані - цифровий підпис, адресу видавця, IPFS-посилання та інформацію про відкликання.

На основі отриманого підпису виконується операція ECDSA-recover, що дозволяє відновити адресу підписанта й перевірити її збіг із адміністративною. За активованої IPFS-перевірки скрипт завантажує файл із контент-адресного сховища ipfs://CID і порівнює його хеш із локально обчисленим значенням.

Фінальний висновок формується автоматично, і результат перевірки вважається успішним лише за умови, що статус має значення official, запис не відкликано, підпис підтверджує адміністративне походження, а хеш даних, отриманих з IPFS, повністю збігається з локальним обчисленням. У такій конфігурації алгоритм унеможливлює ситуацію, за якої змінена, скомпрометована або відкликана прошивка може бути помилково ідентифікована як дійсна.

```
(venv) PS X:\programming\JS\ethereum\diploma> python .\verify_firmware.py .\sketch_dec3a.ino.with_bootloader.bin --check-ipf
s
=== LOCAL FILE ===
Path     : X:\programming\JS\ethereum\diploma\sketch_dec3a.ino.with_bootloader.bin
Size     : 261406 bytes
SHA256   : 0x506a2ce8ba568e377f88c3a74ec5f50901610502889f9f9b95b04b71564cefed

=== NETWORK ===
RPC      : http://127.0.0.1:7545
Chain ID : 1337
Block    : 2
EIP-1559 : yes (baseFee present)

=== CONTRACT ===
Address  : 0x5DB92A6f35b856b8CcCD0678F7ee8b1eC55C12f1

Revoked  : False

=== ON-CHAIN METADATA ===
Name     : FW-1.0
Serial   : UNO-001
Publisher: 0xf830d253C94c9aFBE58043DdfE0A944Ba471fa8D
UploadTS : 1764797231 (2025-12-03 23:27:11)
IPFS     : ipfs://Qmb2fxhmS51UCgZxzN8g5iX3w5t9aRUSiUiXXSGKg49pLh
Admin==Publ?: YES
=== SIGNATURE ===
Sig(hex) : 0xb16a652eb2dc51da391c66aca21df5771ea39e4e7f983041bf7541c0e305570b61f029f9ac120905ee69ac467539450353c5d82e5849906
61269ec7f60e120b91c
r        : 0xb16a652eb2dc51da391c66aca21df5771ea39e4e7f983041bf7541c0e305570b
s        : 0x61f029f9ac120905ee69ac467539450353c5d82e584990661269ec7f60e120b9
v(raw)   : 28
v(27/28) : 28
v(0/1)   : 1
Signer   : 0xf830d253C94c9aFBE58043DdfE0A944Ba471fa8D
Signer==Admin?: YES

=== IPFS CHECK ===
Gateway  : http://127.0.0.1:8080/ipfs/
HTTP URL : http://127.0.0.1:8080/ipfs/Qmb2fxhmS51UCgZxzN8g5iX3w5t9aRUSiUiXXSGKg49pLh
SHA256(IPFS): 0x506a2ce8ba568e377f88c3a74ec5f50901610502889f9f9b95b04b71564cefed
IPFS==Local?: YES

=== RESULT ===
Revoked? : NO
Final OK : YES
```

Рисунок 3.15 – Вивід перевірки зареєстрованої прошивки

На рис. 3.15 продемонстровано результати успішної перевірки, у межах якої хеш офіційної прошивки знайдено у реєстрі, підпис верифіковано як справжній, а вміст, отриманий з IPFS, повністю збігається з локальним бінарним файлом. Такий результат підтверджує ефективність та коректність роботи всієї архітектури системи.

3.6 Практичний сценарій взаємодії з реєстром прошивок

Для демонстрації повного життєвого циклу прошивки було проведено експеримент, який охоплює три різні стани: official, not found та revoked. Усі ці стани є невід’ємною частиною механізму контрольованого розповсюдження прошивок у системі безпеки IoT.

Після виконання команди client.py register виконання алгоритму перевірки засвідчує:

- Status: official;
- Signer==Admin?: YES;
- IPFS==Local?: YES.

Усі ці значення засвідчують коректність публікації прошивки, оскільки її хеш наявний у блокчейні, підпис підтверджує використання адміністративного ключа, а вміст, отриманий з IPFS, повністю відповідає локальному файлу. Відповідний результат подано на рис. 3.15.

Далі було внесено цілеспрямовану модифікацію у вихідний код прошивки. Зміна рядків 23–26 (рис. 3.16) призвела до генерації нового бінарного файлу, а отже - нового SHA-256 хешу.

```
22
23  const char* FW_TAG = "FW-1.0.1";
24  void setup() {
25    Serial.begin(115200);
26    Serial.println(FW_TAG);
27    pinMode(CLK, OUTPUT);
28    pinMode(DIN, OUTPUT);
29    pinMode(CS, OUTPUT);
30
```

Рисунок 3.16 – Модифікація прошивки

Хоча зміни вносилися мінімальні, нова версія прошивки вже не відповідає тому хешу, який зафіксований у реєстрі. Після повторної перевірки система коректно повертає статус "not found", як показано на рис. 3.17.

```
(venv) PS X:\programming\JS\ethereum\diploma> python .\verify_firmware.py .\sketch_dec3a.ino.with_bootloader.bin --check-ipf
s
=== LOCAL FILE ===
Path     : X:\programming\JS\ethereum\diploma\sketch_dec3a.ino.with_bootloader.bin
Size     : 261406 bytes
SHA256   : 0x96b77675d50c6b4cda96826e6a7987b11ab338e553f597b4143b59862c6c1dde

=== NETWORK ===
RPC      : http://127.0.0.1:7545
Chain ID : 1337
Block    : 2
EIP-1559 : yes (baseFee present)

=== CONTRACT ===
Address  : 0x5DB92A6f35b856b8CcCD0678F7ee8b1eC55C12f1
Admin    : 0xf830d253C94c9aFBE58043DdfE0A944Ba471fa8D

=== LOOKUP BY HASH ===
Status   : not found
Result   : FAIL (not found)
```

Рисунок 3.17 - Вивід при перевірці модифікованої прошивки

Поданий результат ілюструє ключову особливість системи, адже навіть мінімальне втручання у вміст прошивки призводить до її автоматичного невизнання реєстром, що однозначно свідчить про її втрату автентичності.
Така поведінка є критичною для захисту IoT-пристроїв від шкідливих втручань.
Для демонстрації повного циклу життєвого управління прошивками було виконано команду відкликання client.py revoke --hash <H>.
Виконання client.py revoke --hash <H> (рис. 3.18) перемикає стан на revoked.

```
(venv) PS X:\programming\JS\ethereum\diploma> python client.py revoke --hash 0x506a2ce8ba568e377f88c3a74ec5f50901610502889f9
f9b95b04b71564cefed
Revoked. tx: 7ed8b96c1dc788807d8916b9bdb7706f28c004008c9833b5be702c09d99d1278
```

Рисунок 3.18 - Запуск команди revoke

Виклик змінює стан прошивки зі «official» на «revoked», що фактично означає примусове блокування її подальшого використання у будь-яких експлуатаційних сценаріях. Такий механізм є критично важливим у випадках виявлення вразливості, дефекту або підтвердженої компрометації конкретної версії, оскільки дозволяє негайно вилучити її з обігу без потреби у видаленні запису з реєстру. На практиці це гарантує, що навіть якщо файл залишиться

доступним у зовнішніх сховищах або у локальних копіях, пристрої, системи оновлення чи засоби аудиту не визнаватимуть його дійсним.

Подальші перевірки, виконані після цього кроку, повертають характерні значення «Status: revoked» та «Final OK: NO», що підтверджує коректну реакцію системи на відкликання та забезпечує чіткий сигнал про заборону використання відповідної прошивки. Такий результат додатково демонструє незмінність і прозорість записів, адже навіть після позначення прошивки як недійсної всі первинні метадані, підпис та IPFS-посилання залишаються доступними для ретроспективного аналізу та аудиту інцидентів. На рис. 3.17 наведено приклад цього процесу, у якому система однозначно визначає прошивку як відкликану та непридатну для встановлення.

```
(venv) PS X:\programming\JS\ethereum\diploma> python .\verify_firmware.py .\sketch_dec3a.ino.with_bootloader.bin --check-ipf
s
=== LOCAL FILE ===
Path     : X:\programming\JS\ethereum\diploma\sketch_dec3a.ino.with_bootloader.bin
Size     : 261406 bytes
SHA256   : 0x506a2ce8ba568e377f88c3a74ec5f50901610502889f9f9b95b04b71564cefed

=== NETWORK ===
RPC      : http://127.0.0.1:7545
Chain ID : 1337
Block    : 3
EIP-1559 : yes (baseFee present)

=== CONTRACT ===
Address  : 0x5DB92A6f35b856b8CcCD0678F7ee8b1eC55C12f1
Admin    : 0xf830d253C94c9aFBE58043DdfE0A944Ba471fa8D

=== LOOKUP BY HASH ===
Status   : revoked
Revoked  : True

=== ON-CHAIN METADATA ===
Name     : FW-1.0
Serial   : UNO-001
Publisher: 0xf830d253C94c9aFBE58043DdfE0A944Ba471fa8D
UploadTS : 1764797231 (2025-12-03 23:27:11)
IPFS     : ipfs://Qmb2fxhmS51UCgZxzN8g5iX3w5t9aRUSiUiXXSGKg49pLh
Admin==Publ?: YES
=== SIGNATURE ===
Sig(hex) : 0xb16a652eb2dc51da391c66aca21df5771ea39e4e7f983041bf7541c0e305570b61f029f9ac120905ee69ac46753945035 3c5d82e584990661269ec7f60e120b91c
r        : 0xb16a652eb2dc51da391c66aca21df5771ea39e4e7f983041bf7541c0e305570b
s        : 0x61f029f9ac120905ee69ac46753945035 3c5d82e584990661269ec7f60e120b9
v(raw)   : 28
v(27/28) : 28
v(0/1)   : 1
Signer   : 0xf830d253C94c9aFBE58043DdfE0A944Ba471fa8D
Signer==Admin?: YES

=== IPFS CHECK ===
Gateway  : http://127.0.0.1:8080/ipfs/
HTTP URL : http://127.0.0.1:8080/ipfs/Qmb2fxhmS51UCgZxzN8g5iX3w5t9aRUSiUiXXSGKg49pLh
SHA256(IPFS): 0x506a2ce8ba568e377f88c3a74ec5f50901610502889f9f9b95b04b71564cefed
IPFS==Local?: YES

=== RESULT ===
Revoked? : YES
Final OK : NO
```

Рисунок 3.19 – Вигляд після запуску команди revoke

Таким чином, система надійно фіксує відгук прошивки, і будь-які спроби використовувати її після відкликання будуть заблоковані механізмом перевірки.

Завершальним етапом демонстрації є аналіз змін балансу адміністратора у Ganache UI. На рис. 3.20 показано, що після серії транзакцій баланс зменшився з 100 ETH до 99 ETH, що підтверджує реальність витрат газу та виконання транзакцій у тестовій мережі.

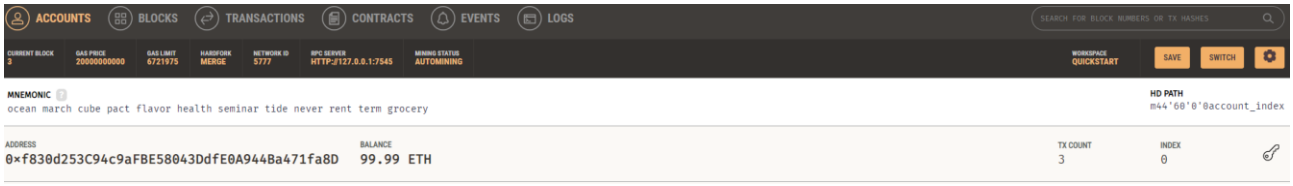


Рисунок 3.20 - Вигляд після усіх транзакцій

Це дозволяє переконатися, що взаємодія зі смарт-контрактом здійснюється відповідно до протоколу Ethereum, а всі транзакції (реєстрація, перевірка, відкликання) виконуються справжніми ончейн-операціями.

3.7 Властивості безпеки, досягнуті реалізацією

Реалізована система забезпечує комплексний набір властивостей безпеки, необхідних для гарантованого контролю над цілісністю та автентичністю прошивок IoT-пристроїв. Насамперед досягається повна криптографічна цілісність, оскільки кожна прошивка ототожнюється зі своїм унікальним SHA-256 хешем, зафіксованим у блокчейні. Навіть незначна зміна одного байта у бінарному файлі призводить до появи нового хешу, який не відповідатиме запису в реєстрі, що, у свою чергу, спричинить повернення статусу «not found».

Така поведінка гарантує, що система миттєво виявляє будь-які модифікації та запобігає використанню неавторизованих версій.

Другим фундаментальним аспектом є автентичність. Підписування хешу прошивки за стандартом EIP-191 та використання алгоритму ECDSA з адміністративним ключем унеможливлює несанкціоновану публікацію артефактів у реєстр. Контракт самостійно відновлює адресу підписанта з переданого підпису і порівнює її з адресою адміністратора. Завдяки цьому система надійно захищена від сторонніх дій осіб, які могли б намагатися зареєструвати фальсифіковану чи шкідливу прошивку.

Третьою важливою властивістю є незаперечність та можливість проведення аудиту. Блокчейн це захищені від підробки та фальсифікації цифрові реєстри, що реалізовані у розподіленому режимі і, як правило, без центрального органу управління. На базовому рівні вони дають змогу спільноті користувачів реєструвати транзакції у спільному реєстрі всередині цієї спільноти, так що за нормального функціонування мережі блокчейну жодна транзакція не може бути змінена після її опублікування [20].

Важливу роль у системі відіграє також контент-адресація через IPFS. Оскільки CID визначається виключно бітовим наповненням файлу, посилання на прошивку однозначно відповідає конкретному її варіанту. Це означає, що навіть якщо IPFS-шлюз, пінінг-сервіс чи файлове сховище будуть підмінені або скомпрометовані, завантажений файл не зможе пройти перевірку, якщо його вміст не збігатиметься з хешем у блокчейні.

Нарешті, система забезпечує механізм оперативного відгуку (revocation). Статус «revoked» дає можливість моментально позначити прошивку як недійсну у разі виявлення вразливості, помилки або компрометації. Усі подальші перевірки одразу засвідчать її недопустимість для встановлення, що дозволяє мінімізувати ризики поширення небезпечних версій у мережі IoT-пристроїв. Завдяки поєднанню цих властивостей система формує надійну інфраструктуру безпеки, здатну протистояти спробам маніпуляції, підміни чи експлуатації прошивок.

ВИСНОВКИ

У ході виконання роботи було спроєктовано та реалізовано комплексну систему забезпечення цілісності й автентичності прошивок IoT-пристроїв на основі поєднання ончейн-реєстру у блокчейні та офчейн-інструментів підписування й верифікації. Запропоноване рішення інтегрує криптографічний контроль бінарного вмісту, механізми цифрового підпису, контент-адресне зберігання у мережі IPFS та незмінність записів блокчейну.

Проведено повний сценарій роботи системи, який охоплює реєстрацію прошивки, перевірку автентичності, виявлення модифікацій та подальше відкликання. У ході експериментів підтверджено, що навіть мінімальне втручання у бінарний файл змінює його SHA-256 хеш, що миттєво фіксується алгоритмом перевірки як стан «not found». Механізм відкликання забезпечує можливість оперативного блокування компрометованих або дефектних версій, а блокчейн гарантує незмінність усіх історичних записів.

Реалізована система демонструє високу надійність завдяки поєднанню трьох незалежних механізмів довіри: криптографічного хешування, цифрового підпису адміністратора та контент-адресації IPFS. Така архітектура забезпечує неможливість підміни прошивки без зміни її ідентифікатора, виключає неоправдані оновлення та формує повний аудит життєвого циклу прошивки.
Досягнута мета роботи, а всі поставлені завдання виконано - розроблено смарт-контракт, створено інструменти публікації та перевірки, проаналізовано поведінку системи в різних сценаріях та підтверджено працездатність механізмів захисту.

Система має практичну цінність і може бути застосована у середовищах, де критично важливо гарантувати довіру до оновлень IoT-пристроїв: промислові контролери, мережеве обладнання, сенсорні мережі, розумні міста.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Жураковський Б.Ю., Зенів І.О. Технології інтернету речей: навч. посібник. – Київ: КПІ ім. Ігоря Сікорського, 2021. – 8 с.

2. Що таке інтернет речей [Електронний ресурс] Режим доступу: https://h7.cl/1l8RJ (дата звернення 27.10.2025).

3. Жураковський Б.Ю., Зенів І.О. Технології інтернету речей: навч. посібник. – Київ: КПІ ім. Ігоря Сікорського, 2021. – 18 с.

4. Яцків Н.Г., Яцків С.В. Перспективи використання технології блокчейн у мережі Інтернет речей // Наук. вісник НЛТУ України. 2016. Вип. 26.8. С. 381.

5. Що таке блокчейн і як він працює? [Електронний ресурс] Режим доступу:https://blog.whitebit.com/uk/what-is-blockchain-technology/ (дата звернення 13.10.2025).

6. Survey of Blockchain-Based Applications for IoT [Електронний ресурс] Режим доступу:https://h7.cl/1gfws (дата звернення 19.10.2025).

7. Інструментальні засоби реалізації вітчизняного стандарту цифрового підпису. [Електронний ресурс] Режим доступу: https://urli.info/1efLP (дата звернення 01.11.2025).

8. What Is a Firmware Update? Why and How to Update Firmware on Your Devices [Електронний ресурс] Режим доступу: https://h7.cl/1l8Rx (дата звернення 22.10.2025).

9. Blockchain-Based Secure Firmware Updates for Electric Vehicle Charging Stations in Web of Things Environments [Електронний ресурс] Режим доступу: https://www.mdpi.com/2032-6653/16/4/226# (дата звернення 24.10.2025).

10. Secure decentralized firmware update delivery service for Internet of Things [Електронний ресурс] Режим доступу: https://h7.cl/1lk54 (дата звернення 27.10.2025).

11. A Highly Secure IoT Firmware Update Mechanism Using Blockchain [Електронний ресурс] Режим доступу: https://www.mdpi.com/1424-8220/22/2/530 (дата звернення 29.10.2025).

12. A Highly Secure IoT Firmware Update Mechanism Using Blockchain [Електронний ресурс] Режим доступу: https://h7.cl/1lk5k (дата звернення 30.10.2025).

13. How to read smart contract data? [Електронний ресурс] Режим доступу: https://www.osl.com/hk-en/academy/article/how-to-read-smart-contract-data (дата звернення 01.11.2025).

14. Rugarc´ıa J., Figueroa-Lorenzo S., Arrizabalaga S., Mohammadzadeh N. Distributed blockchain-based firmware update architecture for IoT environments // Computer Science & Information Technology (CS & IT). – 2024. – P. 102

15. Yaga D., Mell P., Roby N., Scarfone K. Blockchain Technology Overview. - National Institute of Standards and Technology, U.S. Department of Commerce, 2018.- P. 46.

16. A Blockchain-Based OCF Firmware Update for IoT Devices [Електронний ресурс] Режим доступу: https://www.mdpi.com/2076-3417/10/19/6744 (дата звернення 02.11.2025).

17. Upgrading smart contracts [Електронний ресурс] Режим доступу: https://docs.openzeppelin.com/contracts/5.x/learn/upgrading-smart-contracts (дата звернення 04.11.2025).

18. IPFS - Content Addressed, Versioned, P2P File System [Електронний ресурс] Режим доступу: https://www.alphaxiv.org/overview/1407.3561v1 (дата звернення 11.11.2025).

19. Blockchain-based decentralized trust management in IoT: systems, requirements and challenges [Електронний ресурс] Режим доступу: https://link.springer.com/article/10.1007/s40747-023-01058-8 (дата звернення 14.11.2025).

20. Yaga D., Mell P., Roby N., Scarfone K. Blockchain Technology Overview. - National Institute of Standards and Technology, U.S. Department of Commerce, 2018.- P. 6.

ДОДАТОК А
Код смарт контратку

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v4.9.0/contracts/utils/cryptography/ECDSA.sol";

/**
 * @title FirmwareRegistry (ECDSA)
 * Зберігає SHA-256 хеші прошивок та метадані.
 * Потребує ECDSA-підпис адміну під bytes32-хеш прошивки(EIP-191
personal_sign).
 */
contract FirmwareRegistry {
    using ECDSA for bytes32;

    address public admin;

    constructor() {
        admin = msg.sender;
    }

    struct Firmware {
        string  firmwareName;
        string  serialNumber;
        bytes32 hash;        // SHA-256 файлу
        bytes   signature;   // ECDSA підпис admin над hash (personal_sign)
        string  ipfsHash;    // "ipfs://<CID>"
        uint256 uploadTime;
        address publisher;   // = admin
        bool    revoked;
    }

    mapping(bytes32 => Firmware) public firmwares;

    modifier onlyAdmin() {
        require(msg.sender == admin, "Only admin");
        _;
    }

    /// @notice Перевірка підпису admin над bytes32-хеш (EIP-191)
```

```
    function _isValidSignature(bytes32 _hash, bytes memory _signature) internal view
returns (bool) {
        require(_signature.length == 65, "bad signature length");
        // "\x19Ethereum Signed Message:\n32" + hash
        bytes32 ethHash = _hash.toEthSignedMessageHash();
        address signer = ethHash.recover(_signature);
        return (signer == admin);
    }

    /// @notice Дозволяється реєструвати тільки новий хеш і лише з валідним
підписом
    function registerFirmware(
        bytes32 _hash,
        bytes memory _signature,
        string memory _firmwareName,
        string memory _serialNumber,
        string memory _ipfsHash
    ) public onlyAdmin {
        require(firmwares[_hash].uploadTime == 0, "already exists");
        require(_isValidSignature(_hash, _signature), "invalid signature");

        firmwares[_hash] = Firmware({
            firmwareName: _firmwareName,
            serialNumber: _serialNumber,
            hash:        _hash,
            signature:   _signature,
            ipfsHash:    _ipfsHash,
            uploadTime:  block.timestamp,
            publisher:   admin,
            revoked:     false
        });
    }

    function revokeFirmware(bytes32 _hash) public onlyAdmin {
        require(firmwares[_hash].uploadTime != 0, "not found");
        firmwares[_hash].revoked = true;
    }

    function getFirmwareStatus(bytes32 _hash) public view returns (string memory) {
        Firmware memory f = firmwares[_hash];
        if (f.uploadTime == 0) return "not found";
        if (f.revoked) return "revoked";
        return "official";
    }
```

```
    function getFirmwareInfo(bytes32 _hash)
        public
        view
        returns (
            string memory firmwareName,
            string memory serialNumber,
            bytes memory signature,
            string memory ipfsHash,
            uint256 uploadTime,
            address publisher,
            bool revoked
        )
    {
        Firmware memory f = firmwares[_hash];
        return (
            f.firmwareName,
            f.serialNumber,
            f.signature,
            f.ipfsHash,
            f.uploadTime,
            f.publisher,
            f.revoked
        );
    }
}
```

ДОДАТОК Б
Код розгортання клієнту

```
// scripts/deploy_with_ethers.ts  (Remix Scripts, ethers v6)
import { deploy } from './ethers-lib'


(async () => {
  const CONTRACT_NAME = "FirmwareRegistry";

  // ---- Налаштування для RPC fallback (Ganache) ----
  const RPC_URL = "http://127.0.0.1:7545";   // ваш Ganache RPC
  const PRIVATE_KEУ =
"0xbba4c07c2e370d429917c1d655321f2404e736a091c2b7494da2082cc7ccd610"; //
тестовий PK з Ganache

// 1) compile
  try { await remix.call("solidity", "compile"); } catch {}

  // 2) ABI/bytecode
  const comp = await remix.call("solidity", "getCompilationResult");
  const all = comp?.data?.contracts || {};
  let abi: any, bytecode: string | undefined;
  for (const file in all) {
    if (all[file][CONTRACT_NAME]) {
      const art = all[file][CONTRACT_NAME];
      abi = art.abi;
      const obj = art.evm?.bytecode?.object;
      if (!obj) throw new Error("Empty bytecode. Recompile the contract.");
      bytecode = obj.startsWith("0x") ? obj : "0x" + obj;
      break;
    }
  }
  if (!abi || !bytecode) throw new Error(`Contract ${CONTRACT_NAME} not found
in compilation output`);

  // 3) provider + signer (v5/v6)
  const isV6 = !!(ethers as any).BrowserProvider;
  let provider: any, signer: any;

  async function makeInjected() {
    const eth = (window as any).ethereum;
    if (!eth) return null;
```

```
  if (isV6) {
    const p = new (ethers as any).BrowserProvider(eth);
    await p.send("eth_requestAccounts", []);
    const s = await p.getSigner();
    return { provider: p, signer: s };
  } else {
    const p = new (ethers as any).providers.Web3Provider(eth);
    await p.send("eth_requestAccounts", []);
    const s = p.getSigner();
    return { provider: p, signer: s };
  }
}

async function makeRpcWithPK() {
  const P = isV6 ? (ethers as any).JsonRpcProvider : (ethers as
any).providers.JsonRpcProvider;
  const p = new P(RPC_URL);
  const w = new (ethers as any).Wallet(PRIVATE_KEУ, p);
  return { provider: p, signer: w };
}

if (!PRIVATE_KEУ) {
  const inj = await makeInjected();
  if (inj) ({ provider, signer } = inj);
  else ({ provider, signer } = await makeRpcWithPK());
} else {
  ({ provider, signer } = await makeRpcWithPK());
}

const addrFrom = await signer.getAddress();
const net = await provider.getNetwork?.();
const chainIdStr = net?.chainId ? String(net.chainId) : "?";
console.log(`signer: ${addrFrom}`);
console.log(`network: ${chainIdStr}`);

// 4) deploy
const Factory = (ethers as any).ContractFactory;
const factory = new Factory(abi, bytecode, signer);
const contract = await factory.deploy({ gasLimit: 3_000_000 }); // VALUE=0
const txHash =
  (contract as any).deploymentTransaction?.()?.hash ||
  (contract as any).deployTransaction?.hash ||
  "";
if (txHash) console.log(`tx sent: ${txHash}`);
```

```
  if (isV6) {
    await (contract as any).waitForDeployment();
    const addr = await (contract as any).getAddress();
    console.log(`FirmwareRegistry deployed at: ${addr}`);
    const rcp = await (contract as any).deploymentTransaction()?.wait();
    if (rcp?.hash) console.log(`receipt: ${rcp.hash}`);
    if (rcp?.gasUsed) console.log(`gasUsed: ${rcp.gasUsed.toString()}`);
  } else {
    await (contract as any).deployed();
    const addr = (contract as any).address;
    console.log(`FirmwareRegistry deployed at: ${addr}`);
    const rcp = await (contract as any).deployTransaction.wait();
    if (rcp?.transactionHash) console.log(`receipt: ${rcp.transactionHash}`);
    if (rcp?.gasUsed) console.log(`gasUsed: ${rcp.gasUsed.toString()}`);
  }
})().catch((e) => {
  // Логувати лише рядок
  console.log(`DEPLOУ ERROR: ${e?.message || String(e)}`);
});
```

ДОДАТОК
КОД РЕЄСТРАЦІЇ ПРОШИВКИ

```
import argparse, json, os, sys, hashlib
from pathlib import Path
from dotenv import load_dotenv
from web3 import Web3
from eth_account import Account
from eth_account.messages import encode_defunct

load_dotenv()
RPC_URL = os.getenv("RPC_URL", "http://127.0.0.1:7545")
ADMIN_PRIVATE_KEУ = os.getenv("ADMIN_PRIVATE_KEУ")
CHAIN_ID = int(os.getenv("CHAIN_ID", "1337"))

ABI = json.loads(Path("FirmwareRegistry.abi.json").read_text(encoding="utf-8"))
if isinstance(ABI, dict) and "abi" in ABI: ABI = ABI["abi"]
CONTRACT_ADDRESS = Path("contract.address").read_text(encoding="utf-
8").strip()

w3 = Web3(Web3.HTTPProvider(RPC_URL))
if not w3.is_connected(): sys.exit(f"cannot connect {RPC_URL}")
c = w3.eth.contract(address=Web3.to_checksum_address(CONTRACT_ADDRESS),
abi=ABI)
```

```
def sha256_file(p: Path) -> str:
    h = hashlib.sha256()
    with p.open("rb") as f:
        for ch in iter(lambda: f.read(1024*1024), b""): h.update(ch)
    return "0x" + h.hexdigest()

def admin_account():
    if not ADMIN_PRIVATE_KEУ: sys.exit("ADMIN_PRIVATE_KEУ not set")
    return w3.eth.account.from_key(ADMIN_PRIVATE_KEУ)

def build_and_send(acct, tx):
    tx["nonce"] = w3.eth.get_transaction_count(acct.address)
    tx["chainId"] = w3.eth.chain_id
    # EIP-1559 or legacy:
    latest = w3.eth.get_block("latest")
    if latest.get("baseFeePerGas") is not None:
        max_priority = getattr(w3.eth, "max_priority_fee", w3.to_wei(2, "gwei"))
        tx["maxPriorityFeePerGas"] = max_priority
        tx["maxFeePerGas"] = latest["baseFeePerGas"] * 2 + max_priority
    else:
        tx["gasPrice"] = w3.eth.gas_price
    tx.setdefault("gas", 3_000_000)
    signed = w3.eth.account.sign_transaction(tx, private_key=acct.key)
    tx_hash = w3.eth.send_raw_transaction(getattr(signed, "rawTransaction",
signed.raw_transaction))
    return w3.eth.wait_for_transaction_receipt(tx_hash)

def cmd_register(args):
    acct = admin_account()
    fpath = Path(args.file)
    h_hex = sha256_file(fpath)                  # "0x...." (64 hex)
    h_bytes32 = Web3.to_bytes(hexstr=h_hex)      # 32 bytes

    # EIP-191 ("personal_sign") поверх bytes32
    msg = encode_defunct(hexstr=h_hex)          # важно: hexstr !
    sig = Account.sign_message(msg, private_key=acct.key).signature  # 65 байт

    tx = c.functions.registerFirmware(
        h_bytes32, sig, args.name, args.serial, args.ipfs
    ).build_transaction({"from": acct.address})
    receipt = build_and_send(acct, tx)
    print("Registered. tx:", receipt.transactionHash.hex())
```

```
def cmd_revoke(args):
    acct = admin_account()
    h_bytes32 = Web3.to_bytes(hexstr=args.hash)
    tx = c.functions.revokeFirmware(h_bytes32).build_transaction({
        "from": acct.address
    })
    receipt = build_and_send(acct, tx)
    print("Revoked. tx:", receipt.transactionHash.hex())



def main():
    p = argparse.ArgumentParser()
    sub = p.add_subparsers(dest="cmd", required=True)

    p_reg = sub.add_parser("register")
    p_reg.add_argument("file")
    p_reg.add_argument("--name", required=True)
    p_reg.add_argument("--serial", required=True)
    p_reg.add_argument("--ipfs", required=True, help="ipfs://<CID>")
    p_rev = sub.add_parser("revoke")
    p_rev.add_argument("--hash", required=True, help="0x + SHA256 hex")
    p_rev.set_defaults(func=cmd_revoke)

    p_reg.set_defaults(func=cmd_register)

    args = p.parse_args()
    args.func(args)

if __name__ == "__main__":
    main()
```

ДОДАТОК В

Код перевірки прошивки

```
#!/usr/bin/env python3
import argparse, hashlib, json, os, sys, requests, datetime
from pathlib import Path
from web3 import Web3
from eth_account import Account
from eth_account.messages import encode_defunct

def sha256_bytes(b: bytes) -> str:
    return "0x" + hashlib.sha256(b).hexdigest()

def sha256_file(p: Path) -> str:
    h = hashlib.sha256()
    with p.open("rb") as f:
        for ch in iter(lambda: f.read(1024*1024), b""): h.update(ch)
    return "0x" + h.hexdigest()

def ipfs_to_http(ipfs_uri: str, gw="http://127.0.0.1:8080/ipfs/"):
    if not ipfs_uri or not ipfs_uri.startswith("ipfs://"):
        return ""
    return gw.rstrip("/") + "/" + ipfs_uri[len("ipfs://"):]

def to_datetime(ts: int) -> str:
    try:
        return datetime.datetime.fromtimestamp(int(ts)).isoformat(sep=" ",
timespec="seconds")
    except Exception:
        return str(ts)

def sig_split(sig: bytes):
    # 65 bytes: r(32) | s(32) | v(1)
    if not isinstance(sig, (bytes, bytearray)) or len(sig) != 65:
        return None
    r = sig[0:32].hex()
    s = sig[32:64].hex()
    v_raw = sig[64]
    # normalize v to 27/28 and 0/1
    v_27_28 = v_raw if v_raw in (27, 28) else (27 + (v_raw % 2))
    v_0_1   = 0 if v_27_28 == 27 else 1
    return {
        "hex": "0x" + sig.hex(),
        "r": "0x" + r,
        "s": "0x" + s,
```

```
        "v_raw": v_raw,
        "v_27_28": v_27_28,
        "v_0_1": v_0_1
    }

def main():
    ap = argparse.ArgumentParser(description="Firmware verifier (verbose)")
    ap.add_argument("file", help="Path to firmware file")
    ap.add_argument("--rpc", default=os.getenv("RPC_URL","http://127.0.0.1:7545"))
    ap.add_argument("--abi", default="FirmwareRegistry.abi.json")
    ap.add_argument("--addr", default="contract.address")
    ap.add_argument("--check-ipfs", action="store_true", help="Fetch file from IPFS
and compare SHA256")
    ap.add_argument("--ipfs-gateway", default="http://127.0.0.1:8080/ipfs/",
help="IPFS HTTP gateway")
    ap.add_argument("--expect-admin", default="", help="Optional expected admin
address (checks equality)")
    ap.add_argument("--json", action="store_true", help="Also print final JSON
summary")
    args = ap.parse_args()

    # ----- load ABI & address
    try:
        raw = json.loads(Path(args.abi).read_text(encoding="utf-8"))
        ABI = raw["abi"] if isinstance(raw, dict) and "abi" in raw else raw
    except Exception as e:
        sys.exit(f"ERROR: cannot read ABI: {e}")

    try:
        contract_addr = Path(args.addr).read_text(encoding="utf-8").strip()
        caddr_cs = Web3.to_checksum_address(contract_addr)
    except Exception as e:
        sys.exit(f"ERROR: cannot read/parse contract.address: {e}")

    # ----- connect RPC
    w3 = Web3(Web3.HTTPProvider(args.rpc))
    if not w3.is_connected():
        sys.exit(f"ERROR: RPC down: {args.rpc}")

    # network info
    chain_id = w3.eth.chain_id
    latest_block = w3.eth.get_block("latest")
    current_block = latest_block.get("number")
    base_fee = latest_block.get("baseFeePerGas", None)
```

```
    c = w3.eth.contract(address=caddr_cs, abi=ABI)

    # ----- local file info
    fw = Path(args.file)
    if not fw.exists():
        sys.exit(f"ERROR: file not found: {fw}")
    size = fw.stat().st_size
    h_local = sha256_file(fw)

    print("=== LOCAL FILE ===")
    print("Path     :", str(fw.resolve()))
    print("Size     :", f"{size} bytes")
    print("SHA256   :", h_local)
    print()

    print("=== NETWORK ===")
    print("RPC      :", args.rpc)
    print("Chain ID :", chain_id)
    print("Block    :", current_block)
    if base_fee is not None:
        print("EIP-1559 :", "yes (baseFee present)")
    else:
        print("EIP-1559 :", "no")
    print()

    print("=== CONTRACT ===")
    print("Address  :", caddr_cs)
    try:
        admin = c.functions.admin().call()
    except Exception as e:
        sys.exit(f"ERROR: cannot call admin(): {e}")
    print("Admin    :", admin)
    if args.expect_admin:
        print("ExpectAdm:", args.expect_admin, "(OK)" if
admin.lower()==args.expect_admin.lower() else "(MISMATCH)")
    print()

    # ----- status & info
    print("=== LOOKUP BY HASH ===")
    status = c.functions.getFirmwareStatus(h_local).call()
    print("Status   :", status)
    if status == "not found":
        print("Result   : FAIL (not found)")
```

```
        if args.json:
            print(json.dumps({"ok": False, "reason": "not_found", "sha256": h_local},
ensure_ascii=False))
        sys.exit(1)

    name, serial, signature, ipfs_uri, ts, publisher, revoked =
c.functions.getFirmwareInfo(h_local).call()
    print("Revoked  :", revoked)
    print()

    print("=== ON-CHAIN METADATA ===")
    print("Name     :", name)
    print("Serial   :", serial)
    print("Publisher:", publisher)
    print("UploadTS :", ts, f"({to_datetime(ts)})")
    print("IPFS     :", ipfs_uri if ipfs_uri else "(empty)")
    print("Admin==Publ?:", "YES" if admin.lower()==publisher.lower() else "NO")
    print()

    # ----- signature checks
    print("=== SIGNATURE ===")
    splitted = sig_split(signature)
    if not splitted:
        print("Signature:", "(missing or invalid length)")
        print("Result   : FAIL (no/invalid signature on-chain)")
        if args.json:
            print(json.dumps({"ok": False, "reason": "no_signature", "sha256": h_local},
ensure_ascii=False))
        sys.exit(1)

    print("Sig(hex) :", splitted["hex"])
    print("r        :", splitted["r"])
    print("s        :", splitted["s"])
    print("v(raw)   :", splitted["v_raw"])
    print("v(27/28) :", splitted["v_27_28"])
    print("v(0/1)   :", splitted["v_0_1"])

    try:
        signer = Account.recover_message(encode_defunct(hexstr=h_local),
signature=signature)
    except Exception as e:
        print("Recover  : ERROR:", e)
        print("Result   : FAIL (signature recovery error)")
        if args.json:
```

```
            print(json.dumps({"ok": False, "reason": "sig_recover_error", "sha256":
h_local}, ensure_ascii=False))
        sys.exit(1)

    print("Signer   :", signer)
    sig_ok = (signer.lower() == admin.lower())
    print("Signer==Admin?:", "YES" if sig_ok else "NO")
    if not sig_ok:
        print("Result   : FAIL (signature signer != admin)")
        if args.json:
            print(json.dumps({"ok": False, "reason": "signer_mismatch", "sha256":
h_local,
                              "signer": signer, "admin": admin}, ensure_ascii=False))
        sys.exit(1)
    print()

    # ----- IPFS check (optional)
    ipfs_ok = None
    h_ipfs = ""
    if args.check_ipfs and ipfs_uri:
        print("=== IPFS CHECK ===")
        http_url = ipfs_to_http(ipfs_uri, gw=args.ipfs_gateway)
        print("Gateway  :", args.ipfs_gateway)
        print("HTTP URL :", http_url)
        try:
            data = requests.get(http_url, timeout=60).content
            h_ipfs = sha256_bytes(data)
            print("SHA256(IPFS):", h_ipfs)
            ipfs_ok = (h_ipfs == h_local)
            print("IPFS==Local?:", "YES" if ipfs_ok else "NO")
        except Exception as e:
            print("IPFS fetch error:", e)
            ipfs_ok = False
        print()

    # ----- final result
    all_ok = (status == "official") and (not revoked) and sig_ok and (ipfs_ok in (None,
True))
    print("=== RESULT ===")
    print("Revoked? :", "YES" if revoked else "NO")
    print("Final OK :", "YES" if all_ok else "NO")

    if args.json:
        out = {
```

```
            "ok": bool(all_ok),
            "sha256": h_local,
            "file": str(fw.resolve()),
            "size": size,
            "contract": caddr_cs,
            "rpc": args.rpc,
            "chainId": chain_id,
            "block": current_block,
            "status": status,
            "revoked": bool(revoked),
            "name": name,
            "serial": serial,
            "ipfs": ipfs_uri,
            "ipfs_equal": ipfs_ok,
            "sha256_ipfs": h_ipfs,
            "publisher": publisher,
            "admin": admin,
            "signature": splitted
        }
        print(json.dumps(out, ensure_ascii=False, indent=2))

    # exit code mirrors result for scripting/CI
    sys.exit(0 if all_ok else 2)

if __name__ == "__main__":
    main()
```

ДОДАТОК Г

ABI структура


```
[
{
"inputs": [],
"stateMutability": "nonpayable",
"type": "constructor"
},
{
"inputs": [],
"name": "admin",
"outputs": [
{
"internalType": "address",
"name": "",
"type": "address"
}
],
"stateMutability": "view",
"type": "function"
},
{
"inputs": [
{
"internalType": "bytes32",
"name": "",
"type": "bytes32"
}
],
"name": "firmwares",
"outputs": [
{
"internalType": "string",
"name": "firmwareName",
"type": "string"
},
{
"internalType": "string",
"name": "serialNumber",
"type": "string"
},
{
"internalType": "bytes32",
"name": "hash",
```

```
"type": "bytes32"
},
{
"internalType": "bytes",
"name": "signature",
"type": "bytes"
},
{
"internalType": "string",
"name": "ipfsHash",
"type": "string"
},
{
"internalType": "uint256",
"name": "uploadTime",
"type": "uint256"
},
{
"internalType": "address",
"name": "publisher",
"type": "address"
},
{
"internalType": "bool",
"name": "revoked",
"type": "bool"
}
],
"stateMutability": "view",
"type": "function"
},
{
"inputs": [
{
"internalType": "bytes32",
"name": "_hash",
"type": "bytes32"
}
],
"name": "getFirmwareInfo",
"outputs": [
{
"internalType": "string",
"name": "firmwareName",
```

```
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "serialNumber",
                "type": "string"
            },
            {
                "internalType": "bytes",
                "name": "signature",
                "type": "bytes"
            },
            {
                "internalType": "string",
                "name": "ipfsHash",
                "type": "string"
            },
            {
                "internalType": "uint256",
                "name": "uploadTime",
                "type": "uint256"
            },
            {
                "internalType": "address",
                "name": "publisher",
                "type": "address"
            },
            {
                "internalType": "bool",
                "name": "revoked",
                "type": "bool"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "bytes32",
                "name": "_hash",
                "type": "bytes32"
            }
        ],
```

```
        "name": "getFirmwareStatus",
        "outputs": [
                {
                        "internalType": "string",
                        "name": "",
                        "type": "string"
                }
        ],
        "stateMutability": "view",
        "type": "function"
},
{
        "inputs": [
                {
                        "internalType": "bytes32",
                        "name": "_hash",
                        "type": "bytes32"
                },
                {
                        "internalType": "bytes",
                        "name": "_signature",
                        "type": "bytes"
                },
                {
                        "internalType": "string",
                        "name": "_firmwareName",
                        "type": "string"
                },
                {
                        "internalType": "string",
                        "name": "_serialNumber",
                        "type": "string"
                },
                {
                        "internalType": "string",
                        "name": "_ipfsHash",
                        "type": "string"
                }
        ],
        "name": "registerFirmware",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
},
```

```
	{
		"inputs": [
			{
				"internalType": "bytes32",
				"name": "_hash",
				"type": "bytes32"
			}
		],
		"name": "revokeFirmware",
		"outputs": [],
		"stateMutability": "nonpayable",
		"type": "function"
	}
]
```

ДОДАТОК Д
Код прошивки

```
#define CLK 13
#define DIN 11
#define CS  10
#define X_SEGMENTS   4
#define У_SEGMENTS   4
#define NUM_SEGMENTS (X_SEGMENTS * У_SEGMENTS)

// a framebuffer to hold the state of the entire matrix of LEDs
// laid out in raster order, with (0, 0) at the top-left
byte fb[8 * NUM_SEGMENTS];


void shiftAll(byte send_to_address, byte send_this_data)
{
  digitalWrite(CS, LOW);
  for (int i = 0; i < NUM_SEGMENTS; i++) {
    shiftOut(DIN, CLK, MSBFIRST, send_to_address);
    shiftOut(DIN, CLK, MSBFIRST, send_this_data);
  }
  digitalWrite(CS, HIGH);
}


void setup() {
  Serial.begin(115200);
  pinMode(CLK, OUTPUT);
  pinMode(DIN, OUTPUT);
  pinMode(CS, OUTPUT);

  // Setup each MAX7219
  shiftAll(0x0f, 0x00); //display test register - test mode off
  shiftAll(0x0b, 0x07); //scan limit register - display digits 0 thru 7
  shiftAll(0x0c, 0x01); //shutdown register - normal operation
  shiftAll(0x0a, 0x0f); //intensity register - max brightness
  shiftAll(0x09, 0x00); //decode mode register - No decode
}


void loop() {
  static int16_t sx1 = 15 << 8, sx2 = sx1, sy1, sy2;
  sx1 = sx1 - (sy1 >> 6);
```

```
sy1 = sy1 + (sx1 >> 6);
sx2 = sx2 - (sy2 >> 5);
sy2 = sy2 + (sx2 >> 5);

static byte travel = 0;
travel--;
byte *dst = fb;
byte output = 0;
int8_t x_offset = (sx1 >> 8) - X_SEGMENTS * 4;
int8_t y_offset = (sx2 >> 8) - У_SEGMENTS * 4;

uint8_t  screenx, screeny, xroot, yroot;
uint16_t xsumsquares, ysumsquares, xnextsquare, ynextsquare;
int8_t   x, y;

// offset the origin in screen space
x = x_offset;
y = y_offset;
ysumsquares = x_offset * x_offset + y * y;
yroot = int(sqrtf(ysumsquares));
ynextsquare = yroot*yroot;

// Quadrant II (top-left)
screeny = У_SEGMENTS * 8;
while (y < 0 && screeny) {
  x = x_offset;
  screenx = X_SEGMENTS * 8;
  xsumsquares = ysumsquares;
  xroot = yroot;
  if (x < 0) {
    xnextsquare = xroot * xroot;
    while (x < 0 && screenx) {
      screenx--;
      output <<= 1;
      output |= ((xroot + travel) & 8) >> 3;
      if (!(screenx & 7))
        *dst++ = output;
      xsumsquares += 2 * x++ + 1;
      if (xsumsquares < xnextsquare)
        xnextsquare -= 2 * xroot-- - 1;
    }
  }
  // Quadrant I (top right)
  if (screenx) {
```

```
    xnextsquare = (xroot + 1) * (xroot + 1);
    while (screenx) {
      screenx--;
      output <<= 1;
      output |= ((xroot + travel) & 8) >> 3;
      if (!(screenx & 7))
        *dst++ = output;
      xsumsquares += 2 * x++ + 1;
      if (xsumsquares >= xnextsquare)
        xnextsquare += 2 * ++xroot + 1;
    }
  }
  ysumsquares += 2 * y++ + 1;
  if (ysumsquares < ynextsquare)
    ynextsquare -= 2 * yroot-- - 1;
  screeny--;
}
// Quadrant III (bottom left)
ynextsquare = (yroot + 1) * (yroot + 1);
while (screeny) {
  x = x_offset;
  screenx = X_SEGMENTS * 8;
  xsumsquares = ysumsquares;
  xroot = yroot;
  if (x < 0) {
    xnextsquare = xroot * xroot;
    while (x < 0 && screenx) {
      screenx--;
      output <<= 1;
      output |= ((xroot + travel) & 8) >> 3;
      if (!(screenx & 7))
        *dst++ = output;
      xsumsquares += 2 * x++ + 1;
      if (xsumsquares < xnextsquare)
        xnextsquare -= 2 * xroot-- - 1;
    }
  }
  // Quadrant IV (bottom right)
  if (screenx) {
    xnextsquare = (xroot + 1) * (xroot + 1);
    while (screenx--) {
      output <<= 1;
      output |= ((xroot + travel) & 8) >> 3;
      if (!(screenx & 7))
```

```
        *dst++ = output;
       xsumsquares += 2 * x++ + 1;
       if (xsumsquares >= xnextsquare)
        xnextsquare += 2 * ++xroot + 1;
      }
    }
    ysumsquares += 2 * y++ + 1;
    if (ysumsquares >= ynextsquare)
     ynextsquare += 2 * ++yroot + 1;
    screeny--;
  }

  show();
}


void set_pixel(uint8_t x, uint8_t y, uint8_t mode) {
  byte *addr = &fb[x / 8 + y * X_SEGMENTS];
  byte mask = 128 >> (x % 8);
  switch (mode) {
    case 0: // clear pixel
      *addr &= ~mask;
      break;
    case 1: // plot pixel
      *addr |= mask;
      break;
    case 2: // XOR pixel
      *addr ^= mask;
      break;
  }
}


void safe_pixel(uint8_t x, uint8_t y, uint8_t mode) {
  if ((x >= X_SEGMENTS * 8) || (y >= У_SEGMENTS * 8))
    return;
  set_pixel(x, y, mode);
}


// turn off every LED in the framebuffer
void clear() {
  byte *addr = fb;
  for (byte i = 0; i < 8 * NUM_SEGMENTS; i++)
```

```
    *addr++ = 0;
}


// send the raster order framebuffer in the correct order
// for the boustrophedon layout of daisy-chained MAX7219s
void show() {
  for (byte row = 0; row < 8; row++) {
    digitalWrite(CS, LOW);
    byte segment = NUM_SEGMENTS;
    while (segment--) {
      byte x = segment % X_SEGMENTS;
      byte y = segment / X_SEGMENTS * 8;
      byte addr = (row + y) * X_SEGMENTS;

      if (segment & X_SEGMENTS) { // odd rows of segments
        shiftOut(DIN, CLK, MSBFIRST, 8 - row);
        shiftOut(DIN, CLK, LSBFIRST, fb[addr + x]);
      } else { // even rows of segments
        shiftOut(DIN, CLK, MSBFIRST, 1 + row);
        shiftOut(DIN, CLK, MSBFIRST, fb[addr - x + X_SEGMENTS - 1]);
      }
    }
    digitalWrite(CS, HIGH);
  }}
```