УДК 004.056.55:004.738.5:004.272
DOI https://doi.org/10.36059/978-966-397-538-2-6

РЕСУРСООЩАДНИЙ ПІДХІД ДО ПОБУДОВИ SEQURE BOOT У ВБУДОВАНИХ СИСТЕМАХ ІНТЕРНЕТУ РЕЧЕЙ

Ph.D. І. Розломій[1][0000-0001-5065-9004], **Ph.D. А. Ярмілко** [2][0000-0003-2062-2694],
С. Науменко[2] [0000-0002-6337-1605]

[1]*Черкаський державний технологічний університет, Україна,*
[2]*Черкаський національний університет імені Богдана Хмельницького, Україна*
EMAIL: inna-roz@ukr.net, a-ja@ukr.net, naumenko.serhii1122@vu.cdu.edu.ua

Анотація. *У роботі представлено ресурсоощадний підхід до побудови механізму Secure Boot, спеціально адаптованого для вбудованих систем Інтернету речей (IoT), що функціонують в умовах обмежених апаратних ресурсів. Запропонована архітектура враховує критичні обмеження щодо обсягу пам'яті, відсутності апаратної підтримки криптографії та необхідності мінімізації енергоспоживання. Класичні моделі безпечного завантаження малопридатні до таких умов, тому розроблено полегшений варіант Secure Boot з використанням полегшених хеш-функцій, зокрема SPONGENT-128/128/8, що забезпечує базовий рівень автентичності та цілісності програмного забезпечення при мінімальних витратах обчислювальних ресурсів. Реалізація на мікроконтролерах STM32 та ESP8266 продемонструвала високу точність виявлення модифікацій прошивки, енергоефективність (менше 11 мкДж на перевірку) і затримку запуску не більше 30 мс. Розглянуто питання безпечного зберігання еталонного хешу, використання цифрового підпису на основі ECDSA та варіанти масштабування під різні апаратні конфігурації. Запропонований підхід є перспективним для інтеграції в медичні, промислові, військові та інші прикладні IoT-рішення, де важливо забезпечити баланс між безпекою, енергоефективністю та швидкодією. У завершенні окреслено напрями подальших досліджень – впровадження обфускації, захищених OTA-оновлень та підтримки новітніх криптографічних примітивів.*

Ключові слова. *IoT; полегшений Secure Boot; вбудоване програмне забезпечення; криптографічний хеш; SPONGENT; енергоефективність*

1. Постановка проблеми

Інтернет речей (IoT) є одним з ключових напрямів цифрової трансформації у XXI столітті, який змінює уявлення про інформаційні технології та системи керування. Замість централізованих обчислювальних ресурсів у центрі уваги опиняються малі, автономні, часто розподілені пристрої, що інтегруються в реальне середовище та забезпечують моніторинг, керування та передачу даних. Такий підхід відкриває широкі перспективи для автоматизації

життєвих сфер – від «розумного дому» до медичних імплантів, військових платформ та інфраструктур критичного значення. У центрі цих змін – вбудовані мікроконтролери, що працюють у реальному часі та використовують спеціалізоване програмне забезпечення. Незважаючи на свою компактність, ці пристрої здійснюють обробку чутливих даних, керують фізичними процесами й можуть мати прямий вплив на безпеку людини або об'єктів [1].

Програмне забезпечення, зашите у вбудованих пристроях, є фактично їхнім «мозком» – воно визначає функціональність, алгоритмічну логіку, методи обміну даними, реагування на зовнішні сигнали та поведінку пристрою в нестандартних ситуаціях [2]. Втрата довіри до цього компонента призводить до ризиків, які виходять далеко за межі втрати даних. Компрометація мікропрограми може відкрити зловмиснику повний контроль над пристроєм: змінити його логіку, відключити захисні механізми, реалізувати приховані комунікаційні канали або сформувати елемент розподіленої бот-мережі. Це робить захист коду завантаження – особливо на початкових етапах запуску – критично важливою проблемою в контексті IoT-безпеки [3].

Початкова фаза завантаження є найбільш вразливою. Саме в цей момент пристрій ще не активував стандартні механізми захисту: немає шифрування трафіку, відсутні апаратні бар'єри доступу, не працює захист пам'яті. Будь-яке втручання в процес початкового завантаження може бути невидимим для користувача та системних моніторів, що робить подальшу компрометацію глибокою і довготривалою. У традиційних обчислювальних системах (ПК, сервери, мобільні пристрої) цю проблему вирішує механізм Secure Boot – концепція, відповідно до якої жоден код не може бути виконаний до його перевірки на цілісність і автентичність [4, 5, 6]. Як правило, це досягається через вбудовані засоби криптографії, цифрові підписи, зберігання еталонних хешів у захищених зонах та апаратне забезпечення (TPM, HSM або SoC з Secure Boot ROM) [7].

Проте ситуація суттєво змінюється, коли йдеться про IoT-пристрої. Більшість з них побудовано на основі недорогих мікроконтролерів (типу STM32F1, ESP8266, AVR, PIC), які не мають апаратної підтримки криптографії, обмежені у доступній пам'яті (кілька десятків або сотень КБ Flash та RAM), живляться від батарей або енергозалежних джерел і працюють в агресивних середовищах [8, 9, 10]. У таких умовах реалізувати класичну архітектуру Secure Boot практично неможливо. Обмеження за розміром коду, відсутність прискорювачів криптографічних обчислень, жорсткі вимоги до енергоспоживання та швидкодії – усе це змушує шукати альтернативні, ресурсоощадні рішення.

Водночас світова тенденція свідчить про стрімке зростання кількості IoT-пристроїв у мережах різного масштабу – від домашніх до промислових. Це створює нові ризики: атаки на прошивки стають усе більш популярними серед кіберзлочинців, адже проникнення в один слабко захищений вузол може

призвести до ланцюгової реакції в усій системі. Саме тому потреба у захищеному завантаженні в IoT є не розкішшю, а суворою необхідністю [11].

У відповідь на ці виклики необхідно розробити новий підхід до Secure Boot, який відповідатиме можливостям типових мікроконтролерів. Такий підхід повинен базуватись на використанні легковагових криптографічних примітивів (зокрема, хеш-функцій та цифрових підписів із низькими обчислювальними витратами), оптимізації структури завантаження, спрощеному зберіганні еталонних значень, і при цьому забезпечувати достатній рівень криптографічної стійкості. Крім того, важливо враховувати сценарії масштабування системи, підтримку віддалених оновлень (OTA), можливість автентифікації кількох образів прошивок та мінімізацію затримок запуску.

2. Аналіз попередніх досліджень

Захист вбудованого програмного забезпечення у пристроях з обмеженими обчислювальними ресурсами є актуальним напрямом досліджень у контексті забезпечення довіри до пристрою ще на етапі його завантаження [12, 13, 14]. Питання довіри до прошивки, яка керує взаємодією з фізичним середовищем, стає критичним не лише у побутових рішеннях, а й у промислових, медичних та військових застосуваннях. Порушення цілісності початкового коду здатне спричинити як несанкціоноване зчитування або модифікацію даних, так і фізичне пошкодження об'єкта керування. У зв'язку з цим концепція Secure Boot, яка полягає в перевірці автентичності та цілісності програмного коду перед передачею управління на наступний рівень завантаження, є важливим елементом підвищення надійності обчислювальних систем [15].

У типовій архітектурі Secure Boot цей процес реалізується через криптографічну перевірку цифрового підпису або хеш-функції, створеної на основі відкритого ключа виробника або адміністратора системи [16]. Протягом останнього десятиліття активні дослідження були спрямовані на побудову надійного довіреного ланцюга завантаження (chain of trust), в якому кожен етап перевіряє наступний, з використанням алгоритмів хешування SHA-2/SHA-3, підпису RSA, ECDSA, EdDSA тощо.

У повнофункціональних комп’ютерних системах ці механізми реалізуються на рівні UEFI, Trusted Platform Module (TPM) або з використанням апаратних безпечних зон, вбудованих у сучасні мікропроцесори [17]. Така інтеграція дозволяє забезпечити високий рівень контролю та гнучкості в управлінні довірою до програмного забезпечення, забезпечуючи формування надійного кореня довіри [18]. Однак у випадку мікроконтролерів класу STM32, ESP8266, ESP32, RISC-V або AVR, класичні рішення є надмірно ресурсомісткими. Бракує як достатнього обсягу оперативної пам’яті, так і підтримки асиметричної криптографії або спеціалізованих апаратних прискорювачів. Крім того, пристрої часто функціонують в енергозалежному або автономному режимі, що виключає застосування алгоритмів із високим енергоспоживанням.

Серед актуальних підходів до адаптації механізму Secure Boot у таких умовах виокремлюють використання легковагових криптографічних примітивів. Особливої уваги набули компактні хеш-функції, зокрема BLAKE2s, SPONGENT, PHOTON, QUARK, які можуть бути реалізовані навіть на 8-бітних мікроконтролерах [19, 20, 21]. Такі алгоритми мають обмежений розмір коду, не вимагають великих обсягів пам'яті, а також демонструють прийнятний рівень криптостійкості для задач початкової перевірки цілісності.

Ще один напрям досліджень – використання полегшених цифрових підписів на основі кривих з короткими ключами. Популярності набули реалізації Ed25519 та варіанти ECDSA з ключами 160–192 біт, які забезпечують компроміс між стійкістю і продуктивністю [22]. З метою додаткового спрощення, у дослідженнях [23, 24, 25] пропонується відмова від підпису на користь одностороннього хешування, що значно знижує обчислювальне навантаження. Водночас, одностороннє хешування не дозволяє підтвердити авторство коду, а лише перевіряє його незмінність. Такий підхід доцільний для систем із фіксованими прошивками, які не передбачають віддалених оновлень.

Деякі дослідження, зокрема [26, 27], пропонують часткове завантаження (partial or staged booting), коли автентифікації підлягають лише критично важливі сегменти коду, а решта – опціонально. Це дозволяє зменшити час перевірки та обсяг обчислень на етапі запуску. Водночас виникає питання, які саме компоненти вважати критичними та як забезпечити перевірку залежностей між ними.

Паралельно досліджується застосування механізмів Secure Element або Trusted Execution Environment (TEE), що дозволяють винести операції перевірки у спеціалізовану захищену область [28]. Проте такі рішення потребують спеціалізованих мікросхем, збільшують вартість пристрою та його енергоспоживання. Тому вони не є придатними для широкого використання у пристроях з наднизькою собівартістю або в сенсорних мережах, де головним є баланс між безпекою, ціною та тривалістю автономної роботи.

Незмінно актуальною залишається проблема вибору такого криптографічного ядра, яке з одного боку забезпечить необхідний рівень захищеності, а з іншого – не перевищить критичних порогів енергоспоживання та затримки запуску [29]. Це особливо важливо для медичних пристроїв (де затримка запуску може впливати на життєві показники) та військових сенсорних платформ (де енергонезалежність є критичною вимогою).

Попри активні дослідження, нині бракує уніфікованих, гнучких архітектур Secure Boot, які б дозволили адаптивно інтегрувати полегшені криптографічні алгоритми у типові IoT-платформи без жертвування продуктивністю або масштабованістю. Потреба у легких, оптимізованих, але безпечних механізмах безпечного старту залишається відкритим викликом у сфері захисту вбудованого ПЗ у мікроконтролерних системах. Саме тому розробка

ресурсоощадного, гнучкого, масштабованого Secure Boot з використанням легковагових примітивів та модульної верифікації є актуальним напрямом наукових досліджень і практичної реалізації в умовах сучасного IoT.

3. Невирішені питання у сфері Secure Boot

Аналіз засвідчує той факт, що, незважаючи на значний науковий прогрес у сфері забезпечення безпечного завантаження вбудованого програмного забезпечення для IoT-пристроїв, низка важливих аспектів залишається недостатньо дослідженою або має фрагментарні рішення, не придатні до практичного впровадження у реальних системах із жорсткими ресурсними обмеженнями.

Передусім, більшість наявних підходів зосереджені на забезпеченні базової цілісності прошивки, і лише одиничні рішення враховують обмеження на енергоспоживання, обсяг коду та час затримки запуску. Застосування навіть легковагових криптографічних примітивів потребує ретельного балансування між рівнем захисту та реальними можливостями мікроконтролерів. У практиці ж налагодження систем захищеного старту часто стикається з труднощами інтеграції: криптографічні бібліотеки вимагають додаткових ресурсів, а налаштування довірених ключів у пристроях без EEPROM або батареєзалежної пам'яті є технічно складним завданням [30, 31].

По-друге, існує проблема відсутності уніфікованого підходу до організації зберігання еталонних хешів або відкритих ключів у нестійкому середовищі. У багатьох випадках реалізації Secure Boot потребують запису «trusted hash» у флеш-пам'ять, однак такі рішення не завжди захищені від фізичного доступу, перепрошивання або зчитування через JTAG/UART-інтерфейси [32]. Пропозиції щодо використання прихованих сегментів пам'яті, захисту відчитування або шифрування області з еталонними значеннями лише частково вирішують проблему і не охоплюють повного життєвого циклу пристрою – від початкового прошивання до оновлення або ротації ключів [33].

Ще одним відкритим питанням є підтримка сценаріїв оновлення прошивки (OTA) із збереженням довіри. У більшості реалізацій Secure Boot система перевіряє лише першу версію прошивки, без врахування подальших оновлень. Це створює вразливість, коли автентифікований початковий код згодом завантажує модифіковану версію без додаткової перевірки. У той час як промислові рішення передбачають ланцюгову перевірку, у контексті малопотужних IoT-платформ така перевірка поки що не має ефективної реалізації з мінімальним накладом.

Крім того, бракує методик багаторівневої верифікації коду, коли компоненти прошивки (bootloader, ядро, бібліотеки, драйвери) перевіряються незалежно або вибірково залежно від критичності. Це могло б забезпечити більшу гнучкість, зменшити тривалість запуску й оптимізувати ресурсоспоживання. Проте розподіл довіри всередині вбудованої прошивки потребує її спеціальної структури, засобів маркування сегментів і методів

незалежної верифікації. Наразі ці питання не мають уніфікованих рішень для платформ з обмеженнями.

Не менш важливою є проблема адаптації запропонованих схем до широкого спектру апаратних конфігурацій – від 8-бітних мікроконтролерів до 32-бітних платформ з FreeRTOS [34]. Уніфіковані легковагові реалізації часто втрачають ефективність при переході на іншу платформу через відмінності у доступних криптографічних бібліотеках, способах доступу до пам'яті та відсутності спільного стандарту для формування прошивки.

Відсутність публічних моделей оцінки ефективності механізмів Secure Boot в умовах обмежених ресурсів (з урахуванням часу, енергоспоживання, обсягу пам'яті та рівня стійкості до атак) також ускладнює порівняння рішень і їх впровадження в критичні галузі. Оскільки не сформовано типові профілі загроз для вбудованих платформ без апаратної підтримки безпеки, то відсутні й критерії для обґрунтованого вибору мінімальних, але достатніх контрзаходів.

Таким чином, актуальним є формування нового, ресурсоощадного підходу до організації механізму Secure Boot, який базується на таких принципах:

- використання полегшених, але криптографічно стійких примітивів;
- мінімізація затримок та енергоспоживання;
- адаптивна архітектура з фазовою або модульною перевіркою коду;
- підтримка оновлень та ротації ключів;
- універсальність реалізації на платформах без апаратної криптографії.

4. Мета статті

Вирішення проблеми безпечного завантаження у вбудованих IoT-пристроях потребує ресурсоощадного підходу, який базується на сучасних, адаптованих до обмежених обчислювальних середовищ рішеннях.

Метою цього дослідження є розробка та експериментальне обґрунтування ефективного полегшеного механізму Secure Boot для захисту вбудованого програмного забезпечення IoT-пристроїв від несанкціонованої модифікації та запуску стороннього коду.

5. Технічні вимоги до полегшеного Secure Boot

Функціонування механізму Secure Boot у пристроях із низьким рівнем апаратних ресурсів потребує переосмислення стандартних вимог до безпечного завантаження та адаптації їх до специфіки вбудованих систем. На відміну від повнофункціональних комп'ютерних платформ, IoT-пристрої часто не мають розвиненої інфраструктури захисту, зокрема енергонезалежної пам'яті великого обсягу, апаратного криптографічного модуля чи захищеного сховища ключів. У таких умовах розробка полегшеного Secure Boot потребує чіткого визначення технічних вимог, які зберігають базові властивості безпечного старту, але залишаються реалізовними в обмеженому середовищі.

При побудові таких систем важливо не лише зменшити обсяг коду перевірки, а й мінімізувати кількість операцій читання і запису у пам'ять,

адаптувати механізм до енергонезалежного циклу живлення, а також гарантувати відсутність критичних затримок під час запуску. Тому механізм полегшеного Secure Boot повинен поєднувати в собі простоту реалізації, криптографічну стійкість і технологічну універсальність, що дозволить інтегрувати його як у нові пристрої, так і у вже наявні IoT-рішення з відкритим завантажувачем.

Для кращого зрозуміння технічних обмежень, за яких має працювати легкий Secure Boot, важливим є аналізи обчислювальних характеристик поширених платформ мікроконтролерів. На рис. 1 представлено порівняльний огляд ROM, RAM, and CPU для трьох репрезентативних архітектур Інтернету речей.

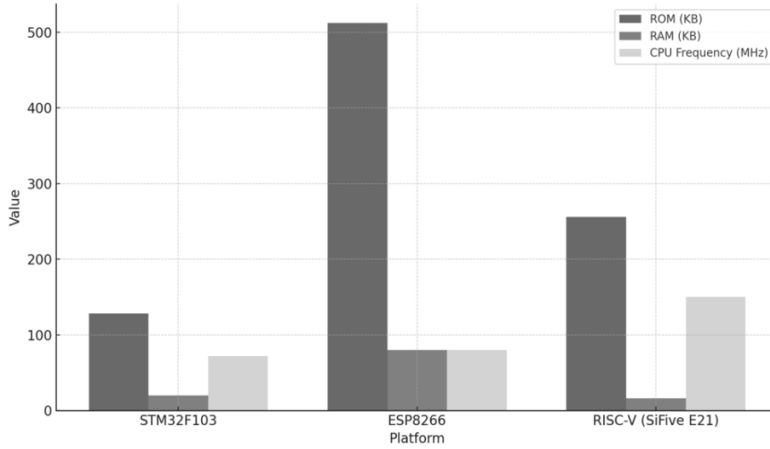

Рисунок 1. Ресурсні обмеження типових IoT-платформ [35]

Як видно з рис. 1, навіть найпотужніші мікроконтролери свого класу, такі як ESP8266, працюють з надто обмеженою оперативною пам'яттю та обчислювальною потужністю, і жодна з перелічених платформ не забезпечує вбудованих апаратних криптографічних механізмів. Ці обмеження вимагають використання легких криптографічних примітивів та мінімальної логіки перевірки в процесі безпечного завантаження.

Передусім, критичною є вимога до забезпечення перевірки цілісності вбудованого програмного забезпечення ще до його запуску. Вона передбачає верифікацію контрольної суми або хешу основної прошивки на основі заздалегідь визначеного еталонного значення. Для цього необхідно забезпечити використання легковагової криптографічної хеш-функції, яка демонструє достатню стійкість до колізій та підробок, але має низьке енергоспоживання та невеликий розмір реалізації. Зокрема, функції

SPONGENT, PHOTON або BLAKE2s у спрощеній формі можуть бути використані у відповідних архітектурах мікроконтролерів.

Зважаючи на специфіку середовища, до хеш-функції також висуваються вимоги щодо мінімального розміру коду (code size), низької затримки обчислення на слабких ядрах (наприклад, ARM Cortex-M3 або Tensilica L106), а також стійкості до сторонніх впливів, таких як помилки живлення або відмови флеш-пам'яті. Крім того, реалізація алгоритму має бути доступною для вбудування в проєкт без залучення складних зовнішніх бібліотек, які можуть перевищити допустимий обсяг прошивки. Тому на практиці ключовим критерієм вибору стає не лише математична надійність, а й придатність для оптимізації, наявність відкритого коду та успішне проходження тестів у типових IoT-середовищах.

Для вибору хеш-функції в умовах обмежених ресурсів необхідно враховувати не лише криптографічну стійкість, але й технічні характеристики реалізації. У табл. 1 наведено порівняння полегшених хеш-функцій, що можуть бути використані у механізмі Secure Boot на мікроконтролерах STM32 та ESP8266.

Таблиця 3

Порівняння полегшених хеш-функцій для застосування в Secure Boot

Хеш-функція	Вимоги до пам'яті (bytes/gates)	Тривалість обчислень	Стійкість до колізій	Енергоспоживання	Підтримка на STM32 / ESP8266
SPONGENT	2260 gates	низька	помірна	дуже низьке	так / частково
PHOTON	2177 gates	середня	висока	низьке	так / експериментально
BLAKE2s	≈ 1300 bytes	висока	дуже висока	середнє	так / оптимізовано

Як видно з табл. 1, SPONGENT має найменше енергоспоживання, однак поступається PHOTON і BLAKE2s у стійкості до колізій. Водночас BLAKE2s забезпечує найвищу криптографічну надійність, але вимагає більше обчислювальних ресурсів.

Другою важливою вимогою є захист від модифікації завантажувача, який сам виконує перевірку основної прошивки. Цей компонент повинен зберігатися у захищеній області пам'яті, за можливості – в енергонезалежній або тільки для читання. У разі відсутності апаратного поділу пам'яті важливо забезпечити принаймні уніфіковану перевірку ланцюга довіри, яка базується на криптографічному зв'язку між завантажувачем і основною прошивкою.

Ще одним ключовим параметром є обсяг доступної пам'яті для реалізації Secure Boot, зокрема кодової та оперативної. У багатьох поширених IoT-

платформах розмір флеш-пам'яті може бути обмежений до 128 або 256 КБ, а обсяг оперативної пам'яті – лише кілька кілобайтів. Тому будь-який алгоритм або структура перевірки має бути реалізована компактно, без значного накладного коду, він не повинен впливати на основну функціональність пристрою. Також необхідно враховувати час виконання перевірки та затримку запуску основної програми. Полегшений Secure Boot повинен забезпечувати мінімальний час початкової перевірки, оскільки у багатьох сенсорних пристроях, особливо в реальному часі, навіть незначна затримка може бути критичною. Баланс між рівнем безпеки та швидкістю запуску має бути визначений експериментально, з урахуванням допустимих часових меж для конкретного застосування. Крім цього, передбачається наявність джерела достовірного еталонного хешу або цифрового підпису, який має бути захищений від підміни. У випадку відсутності апаратного сховища ключів, як у Trusted Platform Module або Secure Element, доцільним є збереження еталонного значення в зашифрованому вигляді або вбудованим безпосередньо в початковий завантажувач.

У практиці проектування захищених IoT-систем одним із підходів є формування хешу прошивки на етапі компіляції з подальшим кодуванням цього значення у внутрішній сегмент пам'яті завантажувача, недоступний для зміни. Таке рішення зменшує потребу в додатковій пам'яті для зберігання ключів і дозволяє реалізувати початкову перевірку без звернення до зовнішніх джерел. Водночас безпечне зберігання ключових елементів у Boot ROM або у hardcoded вигляді потребує додаткового захисту від зчитування, наприклад, шляхом відключення інтерфейсів відладки або використання апаратної фіксації конфігурацій.

На рис. 2 представлено розширену архітектурну модель механізму Secure Boot у мікроконтролерах із обмеженими ресурсами. Ця модель включає не лише базове порівняння хешів, але й безпечне зберігання, генерацію випадкових чисел та посилання на відкритий ключ, що відображає практичні реалізації в реальних пристроях IoT.

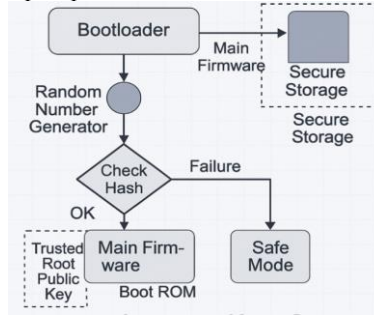

Рисунок 2. Мінімальні компоненти Secure Boot для мікроконтролера

Включення до структури моделі безпечного сховища хешів та довірених ключів дозволяє перевіряти автентичність без використання важких криптографічних модулів, тоді як резервний шлях забезпечує стійкість у разі втручання або невідповідності хешів.

Додатковою перевагою запропонованої архітектури є наявність аварійного режиму (Safe Mode), який дозволяє уникнути повної втрати працездатності пристрою у випадку виявлення некоректного коду. Це важливо для вбудованих систем, що виконують критичні функції, адже збереження мінімального рівня керованості навіть у разі порушення цілісності прошивки підвищує загальну стійкість системи. Саме таке багаторівневе проектування дозволяє досягти балансу між безпекою і витратами ресурсів, що є ключовим для low-power IoT-рішень.

Полегшений Secure Boot має задовольняти сукупність вимог, які забезпечують базову криптографічну перевірку автентичності програмного забезпечення, зберігаючи працездатність системи у межах обмежених обчислювальних, енергетичних та часових ресурсів.

6. Запропонована архітектура полегшеного механізму Secure Boot

У запропонованому підході Secure Boot реалізується як полегшена багатофазна перевірка цілісності та автентичності вбудованого програмного забезпечення, орієнтована на IoT-пристрої з обмеженими обчислювальними ресурсами. Архітектура враховує вимоги до мінімального обсягу пам’яті, обмеженої обчислювальної потужності та швидкодії системи, забезпечуючи при цьому базовий рівень безпеки без потреби в апаратних криптографічних модулях.

6.1. Загальна схема роботи

Механізм Secure Boot у запропонованій архітектурі реалізується як послідовність етапів перевірки цілісності основної прошивки перед її запуском. Його завдання – гарантувати, що на виконання буде передано лише автентичне та незмінене програмне забезпечення. Відразу після подачі живлення або перезавантаження пристрою, система ініціалізує Bootloader, який є єдиною точкою входу в систему до виконання основної програми.

Особливістю реалізації є те, що перевірка виконується в межах мінімально можливої кількості кроків, що знижує загальну затримку старту системи до прийнятного рівня навіть у пристроях з обмеженим тактовим генератором або частотою до 80 МГц. Механізм не передбачає складної структури переходів між фазами, а функції контролю винесені у початкову ділянку коду, що дозволяє жорстко контролювати потік виконання. У випадках, коли пристрій використовує FreeRTOS або подібне середовище, верифікація завершується ще до запуску планувальника. Тому забезпечується чітка логічна межа між захищеним запуском і звичайним виконанням коду.

Bootloader звертається до захищеної області пам’яті, де зберігається еталонний хеш основної прошивки. Паралельно здійснюється обчислення хешу актуального вмісту прошивки з використанням полегшеної хеш-функції.

Далі система виконує порівняння еталонного та розрахованого хешів. У разі збігу контрольна передача переходить до основного коду прошивки. У протилежному випадку пристрій переводиться у режим блокування (Safe Mode) або переходить у резервний сценарій.

На рис. 3 наведено загальну логіку Secure Boot із позначенням ключових вузлів і потоків обробки. Після завершення етапу верифікації система переходить до виконання основного функціонального коду.

Як видно з рис. 3, запропонована структура реалізує багатофазний підхід, у якому ключові перевірки виконуються ще до запуску основного коду. Завдяки використанню лише легковагових криптографічних примітивів і перевірці лише критичних ділянок пам'яті (таких як таблиця переривань, блоки ініціалізації або функції обробки даних), система мінімізує обчислювальні навантаження. Додаткові компоненти, як-от генератор випадкових чисел, відкритий ключ і захищене сховище, активуються лише за необхідності й не задіяні в основному циклі перевірки, що знижує споживання енергії. Такий підхід дозволяє гнучко масштабувати архітектуру відповідно до можливостей конкретної апаратної платформи, зберігаючи при цьому базову перевірку автентичності та цілісності. Завдяки цьому досягається ефективний баланс між швидкодією запуску, енергетичною доцільністю та відповідністю обмеженим ресурсам типових IoT-пристроїв.

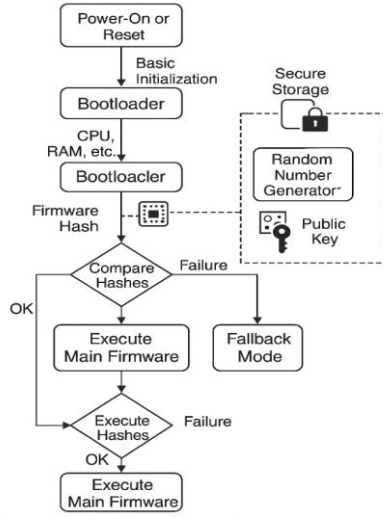

Рисунок 3. Загальний робочий процес запропонованого легкого механізму Secure Boot

6.2. Використані криптографічні елементи

З огляду на обмежену обчислювальну потужність цільових пристроїв, в якості основного криптографічного елемента обрано хеш-функцію SPONGENT-128/128/8, яка забезпечує баланс між стійкістю до атак і компактністю реалізації. Ця функція побудована на основі sponge-конструкції, що складається з раундів перестановок та побітових операцій, і дозволяє формувати хеш-підпис зі змінною довжиною на основі обмежених вхідних блоків. Характерною особливістю SPONGENT є її здатність ефективно працювати навіть на мікроконтролерах із частотою менше 100 МГц, споживаючи мінімум енергії (менше 10 μJ на операцію). Алгоритм хешування можна представлений, як (1).

$$Z = Sponge[f, pad, r](M), \quad (1)$$

де f – функція внутрішнього перестановки (у SPONGENT – побудована на основі π-раундів), pad – алгоритм доповнення до потрібної довжини, $r=8$ – кількість бітів, які обробляються за раунд, M – вхідне повідомлення (вміст прошивки або її критичні ділянки), Z – вихідний хеш.

Послідовність обробки виглядає так:

1. Ініціалізація внутрішнього стану: $S_0 = IV$.
2. Раунди поглинання: $S_{i+1} = f(S_i \oplus m_i)$ для кожного блоку m_i.
3. Формування результату: перші n бітів з фінального стану $Z = first_n(S_k)$.

Цей підхід дозволяє отримувати хеш зі змінною довжиною без потреби у використанні складних або енергозатратних операцій. У критичних пристроях реального часу SPONGENT дає змогу забезпечити перевірку цілісності з мінімальною затримкою старту системи.

У тих випадках, коли архітектура пристрою допускає використання більш складних алгоритмів, може застосовуватись BLAKE2s, який має вищу криптографічну надійність та перевірену реалізацію в низці безпекових бібліотек. Також може бути використано PHOTON, який демонструє стійкість до атак диференціального та лінійного криптоаналізу й має компактну реалізацію для платформ із пам'яттю до 8 КБ.

У базовому варіанті архітектури механізм Secure Boot не використовує цифровий підпис, що дозволяє зменшити обсяг коду, кількість операцій із великими числами та час затримки запуску. Однак у випадках, де важлива автентифікація джерела прошивки, передбачено розширений варіант з використанням полегшеного цифрового підпису на основі ECDSA з 160-бітовими кривими (наприклад, secp160r1). Такий підпис потребує мінімум двох еліптичних множень під час перевірки, однак здатен гарантувати достовірність навіть у разі доступу з боку атакуючого до зовнішньої пам'яті.

Процедура перевірки підпису у цьому випадку виконується таким чином:

Обчислюється хеш повідомлення (прошивки): $e = HASH(M)$.

Визначаються допоміжні значення:

$\varpi = s^{-1} \bmod n,\ u_1 = e \cdot \varpi \bmod n, u_2 = r \cdot \varpi \bmod n.$

Обчислюється точка на кривій: $P = u_1 G + u_2 QP$, де G – базова точка, Q – відкритий ключ, r, s_r – параметри підпису.

Перевірка автентичності: $r \equiv x_p \bmod n.$

Тут x_p – x-координата точки P, що отримана в результаті обчислень. Усі еліптичні операції реалізуються за допомогою арифметики, а відкрита частина ключа може бути збережена в ROM, вбудована у завантажувач або зашифровано витягнута з зовнішнього джерела.

Автентичність хешу підтверджується відкритим ключем, який може бути збережений кількома способами:

- у енергонезалежній ROM-пам'яті (захищений варіант);
- у початковому завантажувачі (жорстко зашитий ключ);
- завантажений з зашифрованого джерела, за умови наявності модуля дешифрування.

Цей підхід дозволяє забезпечити односторонній довірчий ланцюг навіть за відсутності повноцінного TPM або Secure Element. Обчислення цифрового підпису або його перевірка активуються лише у фазі оновлення або перевірки еталонного хешу, не зачіпаючи основний цикл запуску, що дає змогу уникати енергетичних піків у критичних режимах роботи.

6.3. Збереження та перевірка коду

Еталонний хеш основної прошивки зберігається у захищеній області флеш-пам'яті, до якої Bootloader має доступ лише в режимі читання. У деяких реалізаціях, де можливе апаратне розділення пам'яті, використовується окремий ROM-сегмент, що забезпечує незмінність еталонного значення.

Особливістю реалізації є перевірка не всієї прошивки, а лише її найбільш критичних ділянок: початкового коду ініціалізації, таблиці переривань, функцій обробки мережевого трафіку. Це зменшує обсяг обчислень і дозволяє виконувати перевірку за прийнятний час. Стратегія запуску базується на логіці fail-stop: у разі виявлення невідповідності виконання основної програми не отримує дозволу, а пристрій переходить у заблокований або резервний режим.

Послідовність дій з перевірки автентичності та цілісності програмного забезпечення включає такі етапи:

1. Після подачі живлення або перезапуску пристрою активується Bootloader, який здійснює базову ініціалізацію системи та перевірку доступу до пам'яті.

2. Визначаються межі критичних ділянок прошивки, що підлягають перевірці. Це можуть бути області з кодом початкової ініціалізації, таблиця векторів переривань або функції, що обробляють мережевий трафік.

3. Bootloader ініціює хешування обраних сегментів з використанням полегшеної хеш-функції (SPONGENT, PHOTON або BLAKE2s), яка адаптована до умов низької продуктивності та обмеженого енергоспоживання.

4. Обчислений хеш порівнюється з еталонним значенням, збереженим у захищеній області пам'яті. Порівняння виконується без проміжних копій у змінній пам'яті, що унеможливлює маніпуляції під час перевірки.

5. У разі успішного збігу Bootloader передає управління основній програмі, яка починає своє виконання згідно із завантаженою логікою.

6. Якщо виявлено розбіжність між обчисленим і еталонним хешем, система негайно зупиняє завантаження та переходить у безпечний стан Safe Mode, з блокуванням інтерфейсів або ініціацією процесу оновлення з довіреного джерела.

7. Для підвищення надійності у деяких реалізаціях допускається зберігання декількох допустимих хешів, що дозволяє підтримувати кілька версій прошивки та реалізовувати механізм резервного запуску.

Запропонована послідовність перевірки дозволяє дотриматися принципів безпеки з урахуванням технічних обмежень платформи та зберегти продуктивність системи навіть в умовах критично низьких ресурсів.

Розроблений механізм забезпечує не лише перевірку цілісності ключових сегментів програмного коду, але й створює довірене середовище для подальшої роботи пристрою. Адаптація класичної моделі Secure Boot до умов обмежених ресурсів дозволяє забезпечити базові гарантії безпеки без суттєвого зростання часу завантаження або обсягу коду. Завдяки модульності та підтримці кількох хешів у пам'яті, механізм легко масштабувати та інтегрувати у гетерогенні IoT-системи, де критично важливими є автономність, енергоефективність та надійність запуску.

7. Експериментальна частина та оцінка ефективності

Для перевірки працездатності запропонованої архітектури Secure Boot було проведено її експериментальну реалізацію на двох типових платформах IoT-класу: STM32F103C8T6 (мікроконтролер на базі ARM Cortex-M3 з тактовою частотою 72 МГц) та ESP8266 NodeMCU (процесор Tensilica L106, 80 МГц). Обидва мікроконтролери широко використовуються в проєктах з обмеженим енергоспоживанням і мають низький обсяг оперативної пам'яті, що дозволяє адекватно оцінити ефективність механізму в умовах обмежених ресурсів.

Було реалізовано прототип Secure Boot, що включав Bootloader обсягом 5,1 КБ, модуль обчислення хешу SPONGENT-128/128/8, захищене збереження еталонного хешу, а також верифікацію перед виконанням основної програми. В якості тестової прошивки використовувалась програма обсягом 32 КБ, у якій навмисно модифікувались критичні ділянки, такі як таблиця переривань та функції обробки даних.

Програмування здійснювалось мовою C з використанням компілятора GCC (для STM32) та Arduino Core (для ESP8266). Параметри часу перевірки, енергоспоживання та обсяг коду оцінювались з використанням STM32CubeMonitor, INA219 (для вимірювання миттєвого струму) та осцилографа Tektronix. У результаті експериментів встановлено, що для

STM32 повний цикл перевірки займає 21,7 мс, для ESP8266 – 29,1 мс. Обчислення хешу SPONGENT тривало 18,2 мс і 25,4 мс відповідно, при цьому енергоспоживання однієї перевірки не перевищувало 10–11 мікроджоулів. Обсяг коду Secure Boot після компіляції склав 6,4 КБ для STM32 та 7,1 КБ для ESP8266.

Під час реалізації особливу увагу було приділено оптимізації обчислювального ядра хеш-функції та мінімізації звернень до пам'яті. Виявлено, що найбільший вплив на час перевірки має кількість обраних для верифікації сегментів, а також глибина циклів обробки даних у SPONGENT. У випадку STM32 було досягнуто стабільної роботи навіть при роботі з малими блоками даних по 64 байти, тоді як на ESP8266 виявлено затримки при роботі з блоками понад 128 байт. Це свідчить про критичну роль оптимального розміру оброблюваного сегмента, що має враховуватись при адаптації механізму до конкретної платформи.

Для оцінки ефективності використовувалися три основні метрики: тривалість перевірки, енергоспоживання під час старту, а також розмір коду завантажувача. Тестові сценарії передбачали як перевірку цілісної прошивки, так і симуляцію модифікованого коду. Симуляція атаки шляхом модифікації коду показала, що механізм коректно виявляє зміну навіть одного байта в перевірюваній області, що підтверджує високу чутливість і надійність алгоритму. У всіх тестах спрацювання fail-safe механізму спостерігалося негайно – з переходом до Safe Mode менш ніж за 1 мс після невдалої перевірки. Таким чином, запропонований механізм забезпечує високу швидкість реакції на загрози без суттєвого навантаження на апаратну частину пристрою. Результати експерименту наведено у табл. 2.

Таблиця 2

Експериментальна оцінка запропонованої реалізації Secure Boot

Метрика	STM32F103C8T6	ESP8266 NodeMCU
Розмір коду завантажувача	6.4 KB	7.1 KB
Тривалість хешування (SPONGENT)	18.2 ms	25.4 ms
Тривалість перевірки	21.7 ms	29.1 ms
Енергоспоживання на перевірку	9.8 µJ	11.3 µJ
Витрати пам'яті (RAM)	< 512 bytes	< 768 bytes
Виявлення втручання	100%	100%
Реакція захисту основної прошивки	Safe Mode або відкат	Safe Mode або відкат

Як показано в таблиці, обидві платформи демонструють високу точність у виявленні зміненої прошивки без помітної затримки запуску системи. Найнижчі значення енергоспоживання та часу перевірки досягнуто завдяки

використанню легковагових криптографічних примітивів та обмеженню перевірки лише до критичних сегментів пам'яті. Це дозволяє ефективно інтегрувати захист в системи з чутливістю до затримки або автономним живленням. Отримані результати підтверджують перспективність запропонованої архітектури як гнучкого рішення для безпечного старту у пристроях з обмеженими ресурсами.

У проведеній серії експериментів система успішно ідентифікувала підміну прошивки: у випадках зміни контрольних байтів пристрій переходив у безпечний режим без запуску основної програми. Це свідчить про здатність механізму протидіяти несанкціонованим модифікаціям навіть у разі фізичного доступу до пам'яті пристрою. Порівняно з традиційними підходами, які потребують цифрового підпису та еліптичної криптографії, запропонований варіант з полегшеним хешуванням показав у 3–4 рази нижчу тривалість запуску та на порядок менше енергоспоживання.

Однак у базовій реалізації не враховано захист еталонного хешу від прямого зчитування, що може бути критичним у разі фізичного злому. Подальші дослідження планується спрямувати на інтеграцію методів обфускації, використання Secure Element, а також впровадження довіреного віддаленого оновлення (OTA) з криптографічною перевіркою на сервері. Отримані результати підтверджують доцільність впровадження механізму в практичні IoT-рішення, де критичним є поєднання безпеки та енергоефективності.

8. Обговорення

Результати експериментів підтвердили, що запропонований механізм полегшеного Secure Boot може бути ефективно реалізований на мікроконтролерах із обмеженими ресурсами, зберігаючи баланс між продуктивністю та базовим рівнем безпеки. Використання полегшеної хеш-функції та обмеження перевірки до критичних сегментів прошивки дозволило досягти прийнятної тривалості запуску та мінімального енергоспоживання, що є надзвичайно важливим для автономних IoT-пристроїв. Успішне виявлення модифікованих прошивок у всіх тестових сценаріях засвідчує здатність запропонованої архітектури протидіяти базовим загрозам без потреби в складних апаратних модулях.

Разом із тим, певні обмеження все ще залишаються актуальними. Зокрема, зберігання еталонного хешу у відкритому вигляді навіть у захищеній флеш-пам'яті залишає простір для потенційних атак у разі фізичного доступу до пристрою. Також у поточній реалізації відсутній захищений механізм оновлення прошивки, що ускладнює використання рішення в динамічному середовищі з частими оновленнями. Не враховано також можливість інтеграції із зовнішніми джерелами довіри або віддаленими серверами перевірки, що є необхідним для побудови повноцінного ланцюга довіри.

Перспективними напрямами подальших досліджень є інтеграція механізмів шифрування або обфускації еталонного хешу, впровадження розширеного варіанту із цифровим підписом, а також використання зовнішніх апаратних компонентів, таких як Secure Element або TPM. Окрема увага має бути приділена оптимізації енергоспоживання в процесі оновлення та розробці архітектури, яка підтримує безпечне віддалене оновлення з верифікацією на сервері. Доцільним є також розширення підтримки новітніх криптографічних алгоритмів, орієнтованих на енергоефективність, та включення інструментів моніторингу поведінки пристрою під час завантаження для побудови адаптивної, самовідновлюваної системи захисту.

9. Висновки та перспективи

У статті запропоновано архітектуру полегшеного механізму Secure Boot, адаптовану до умов обмежених ресурсів IoT-пристроїв. Розроблений підхід ґрунтується на використанні легковагових криптографічних примітивів, зокрема хеш-функції SPONGENT-128/128/8, та передбачає перевірку лише критичних ділянок прошивки з метою мінімізації енергоспоживання та часу запуску. Запропоноване рішення не вимагає наявності апаратних криптомодулів і може бути реалізоване на популярних мікроконтролерах, таких як STM32 та ESP8266. Архітектура зберігає логіку fail-stop і дозволяє уникати запуску зміненого коду без необхідності у складному апаратному захисті. Водночас, її реалізація потребує мінімального обсягу пам’яті, що дозволяє використовувати її у вкрай ресурсно обмежених пристроях.

Експериментальна реалізація підтвердила працездатність і ефективність розробленого механізму: затримка перевірки становила менше 30 мс, а енергоспоживання залишалося в межах 10–11 мікроджоулів. Усі модифікації прошивки були успішно виявлені, що свідчить про доцільність застосування запропонованого підходу в системах, де критичними є одночасно безпека та енергоефективність. Прототип функціонував стабільно при багаторазових перезапусках, демонструючи низький ризик збоїв навіть за умов коливань напруги. Це особливо важливо для IoT-рішень, які працюють у нестабільному середовищі або на живленні від батарей.

Результати дослідження відкривають перспективу подальшого вдосконалення архітектури, зокрема впровадження підтримки цифрового підпису в розширеній конфігурації, обфускації еталонних хешів та інтеграції із захищеним модулем оновлення програмного забезпечення. Запропоноване рішення є гнучким і придатним для масштабування на інші класи пристроїв, що робить його потенційною основою для створення довірених IoT-середовищ.

Література

1. Rozlomii, I., Yarmilko, A., & Naumenko, S. (2025). Innovative resource-saving security strategies for IoT devices. *Journal of Edge Computing*, *4*(1), 35–56.

2. Yarmilko, A., Rozlomii, I., & Naumenko, S. (2024, May). Dependability of Embedded Systems in the Industrial Internet of Things: Information Security and Reliability of the Communication Cluster. In *International Scientific-Practical Conference" Information Technology for Education, Science and Technics"* (pp. 235-249). Cham: Springer Nature Switzerland.

3. Qasem, A., Shirani, P., Debbabi, M., Wang, L., Lebel, B., & Agba, B. L. (2021). Automatic vulnerability detection in embedded devices and firmware: Survey and layered taxonomies. *ACM Computing Surveys (CSUR)*, *54*(2), 1-42.

4. Wang, R., & Yan, Y. (2022, February). A survey of secure boot schemes for embedded devices. In *2022 24th International Conference on Advanced Communication Technology (ICACT)* (pp. 224-227). IEEE.

5. Cano-Quiveu, G., Ruiz-de-Clavijo-Vazquez, P., Bellido, M. J., Juan-Chico, J., & Viejo-Cortes, J. (2023). IRIS: An embedded secure boot for IoT devices. *Internet of Things*, *23*, 100874.

6. de Clavijo Vázquez, P. R., Bellido, M. J., Chico, J. J., Cortes, J. V., & Vallecillo, J. B. (2024). Hardware Secure Boot. A Review Of" IRIS: An Embedded Secure Boot for IoT devices". *IX Jornadas Nacionales de Investigación En Ciberseguridad*, 512-513.

7. Faure, E., Rozlomii, I., & Naumenko, S. (2025). Cryptographic load sharing method in critical infrastructure sensor networks. In *Proceedings of the Fourth International Conference on Cyber Hygiene & Conflict Management in Global Information Networks (CH&CMiGIN'25)* (pp. 5–15).

8. Dhrubo, A. F. H., Hasan, S., & Qayum, M. A. (2025). STM32-Based IoT Framework for Real-Time Environmental Monitoring and Wireless Node Synchronization. *arXiv preprint arXiv:2506.17295*.

9. Ahemd, A., & Badawy, W. (2023). Design of IoT Wireless Programmer for Microchip AVR Microcontrollers' OTA Firmware Updates. DOI: 10.21203/rs.3.rs-3504697/v1.

10. Vdovichenko, O., & Perepelitsyn, A. (2024). Analysis of product range of microcontrollers for creation of elements of home automation. *Aerospace Technic and Technology*, (5), 118-126.

11. Rozlomii, I., Yarmilko, A., Naumenko, S., & Mykhailovskyi, P. (2024, September). Hardware encryptors and cryptographic libraries for optimizing security in IoT. In *Proceedings of the 12th International Conference Information Control Systems & Technologies (ICST 2024), Odesa, Ukraine* (pp. 99-109).

12. Kornaros, G. (2022). Hardware-assisted machine learning in resource-constrained IoT environments for security: review and future prospective. *IEEE Access*, *10*, 58603-58622.

13. De Oliveira Nunes, I., Jakkamsetti, S., Kim, Y., & Tsudik, G. (2022, October). Casu: Compromise avoidance via secure update for low-end embedded systems. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design* (pp. 1-9).

14. Grisafi, M., Ammar, M., Roveri, M., & Crispo, B. (2022). {PISTIS}: Trusted computing architecture for low-end embedded systems. In *31st USENIX Security Symposium (USENIX Security 22)* (pp. 3843-3860).

15. Arunkumar, S. (2024, May). Implementation and Verification of Secure Boot in Embedded Systems. In *2024 International Conference on Advances in Modern Age Technologies for Health and Engineering Science (AMATHE)* (pp. 1-4). IEEE.

16. Jyothi, T., & Jain, S. (2023, May). Tpm based secure boot in embedded systems. In *2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC)* (pp. 786-790). IEEE.

17. Schmidt, S., Tausig, M., Koschuch, M., Sheng, B., Hudler, M., Simhandl, G., ... & Michael, M. K. (2025). Leveraging Trusted Platform Modules (TPM) for Cryptographic Anchoring and Remote Attestation of UEFI Capsule Updates in Secure Boot Environments.

18. Ling, Z., Yan, H., Shao, X., Luo, J., Xu, Y., Pearson, B., & Fu, X. (2021). Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes. *Journal of Systems Architecture*, *119*, 102240.

19. Tandel, P., & Nasriwala, J. (2024). Efficient authentication framework with Blake2s and a hash-based signature scheme for Industry 4.0 applications. *International Journal of Intelligent Engineering Informatics*, *12*(4), 410-432.

20. Hiremath, S. B., Heblikar, S., Javali, M. S., Tejas, M., & Iyer, N. C. (2024, August). Side Channel Analysis and Hardware Implementation of AES CBC Algorithm on Artix A7 FPGA Development Boards. In *International Conference on ICT for Sustainable Development* (pp. 111-120). Singapore: Springer Nature Singapore.

21. Al-Shatari, M., Hussin, F. A., Aziz, A. A., Eisa, T. A. E., & Tran, X. T. (2023). IoT Edge Device Security: An Efficient Lightweight Authenticated Encryption Scheme Based on LED and PHOTON. *Applied Sciences*, *13*(18), 10345.

22. Brendel, J., Cremers, C., Jackson, D., & Zhao, M. (2021, May). The provable security of ed25519: theory and practice. In *2021 IEEE Symposium on Security and Privacy (SP)* (pp. 1659-1676). IEEE.

23. Román, R., Arjona, R., & Baturone, I. (2023). A lightweight remote attestation using PUFs and hash-based signatures for low-end IoT devices. *Future Generation Computer Systems*, *148*, 425-435.

24. Songshen, H. A. N., Kaiyong, X. U., Zhiqiang, Z. H. U., Songhui, G. U. O., Haidong, L. I. U., & Zuohui, L. I. (2022). Hash-based signature for flexibility authentication of IoT devices. *Wuhan University Journal of Natural Sciences*, *27*(1), 1-10.

25. Chakraborty, A., & Biswas, A. (2025, February). A Comprehensive and Systematic Analysis of One-Way Hashing and Its Advancements. In *2025 3rd International Conference on Intelligent Systems, Advanced Computing and Communication (ISACC)* (pp. 1309-1316). IEEE.

26. Li, B., & Dong, W. (2021). Edge-centric programming for iot applications with automatic code partitioning. *IEEE Transactions on Computers*, *71*(10), 2408-2422.

27. Dong, Y., Duan, S., Lyu, F., Zhao, P., Zhang, Y., Ren, J., & Zhang, Y. (2024). NC-Load: On-Demand Program Loading and Running for Computing Sharing Among IoT Devices. *IEEE Internet of Things Journal*.

28. Muñoz, A., Ríos, R., Román, R., & López, J. (2023). A survey on the (in) security of trusted execution environments. *Computers & Security*, *129*, 103180.

29. Rozlomii, I., Naumenko, S., Mykhailovskyi, P., & Monarkh, V. (2024, October). Resource-Saving Cryptography for Microcontrollers in Biomedical Devices. In *2024 IEEE 5th KhPI Week on Advanced Technology (KhPIWeek)* (pp. 1-5). IEEE.

30. Bruno, G. (2023). *Design and Implementation of Trusted Channels in the Keystone Framework* (Doctoral dissertation, Politecnico di Torino). URL: https://webthesis.biblio.polito.it/29457/.

31. Rowland, M., & Karch, B. (2022). *A Review of Technologies that can Provide a 'Root of Trust' for Operational Technologies*. URL: https://www.osti.gov/servlets/purl/1861944.

32. Kumar, O. (2024). *Embedded Firmware Debugging and Telemetry* (Master thesis, TU Delft). URL: https://repository.tudelft.nl/record/uuid:f3b05137-f9aa-49c9-bee9-6797a742b14f.

33. Duan, X., Li, J., Pei, X., Ergesh, T., Wen, Z., & Chen, M. (2022, August). Implementation of CASPAR Firmware Interface on PYNQ-Z2. In *2022 IEEE 9th International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications (MAPE)* (pp. 53-57). IEEE.

34. He, C., Zhang, J., Cai, Y., Zi, C., & Jiang, X. (2021). An implementation of module management controller for MicroTCA data processing system. *Journal of Instrumentation*, *16*(03), T03005.

35. Thakor, V. A., Razzaque, M. A., & Khandaker, M. R. (2021). Lightweight cryptography algorithms for resource-constrained IoT devices: A review, comparison and research opportunities. *IEEE Access*, *9*, 28177-28193.

RESOURCE-EFFICIENT APPROACH TO SEQURE BOOT IN EMBEDDED INTERNET OF THINGS SYSTEMS

Ph.D. I. Rozlomii[1][0000-0001-5065-9004], **Ph.D. A. Yarmilko**[2][0000-0003-2062-2694], **S. Naumenko**[2][0000-0002-6337-1605]

[1]*Cherkasy State Technological University, Ukraine*
[2]*Bohdan Khmelnytsky National University of Cherkasy, Ukraine*
EMAIL: inna-roz@ukr.net, a-ja@ukr.net, naumenko.serhii1122@vu.cdu.edu.ua

Abstract. *This work presents a resource-efficient approach to the design of a Secure Boot mechanism specifically tailored for embedded Internet of Things (IoT) systems operating under constrained hardware resources. The proposed*

architecture addresses critical limitations such as restricted memory capacity, lack of hardware cryptographic support, and the necessity to minimize energy consumption. Classical secure boot models are largely unsuitable for these conditions; therefore, a lightweight Secure Boot variant has been developed using lightweight hash functions, in particular SPONGENT-128/128/8, which ensures a basic level of software authenticity and integrity at minimal computational cost. Implementation on STM32 and ESP8266 microcontrollers demonstrated high accuracy in detecting firmware modifications, energy efficiency (less than 11 μJ per verification), and a startup delay not exceeding 30 ms. The study discusses secure storage of the reference hash, the use of ECDSA-based digital signatures, and scaling options for different hardware configurations. The proposed approach is promising for integration into medical, industrial, military, and other applied IoT solutions where balancing security, energy efficiency, and performance is crucial. The conclusion outlines directions for future research, including obfuscation techniques, secure OTA updates, and support for emerging cryptographic primitives.

Keywords. *IoT; lightweight Secure Boot; embedded software; cryptographic hash; SPONGENT; energy efficiency*

UDC 004.8
DOI https://doi.org/10.36059/978-966-397-538-2-7

METHOD FOR ANALYZING THE UKRAINIAN LANGUAGE TEXTS SENTIMENT USING NATURAL LANGUAGE PROCESSING

O. Zalutska[0000-0003-1242-3548]
Khmelnytskyi National University, Ukraine
EMAIL: zalutsk.olha@gmail.com

Abstract. The paper focuses on intelligent sentiment analysis of text related to named entities. The proposed method combines a neural network-based natural language processing model, a lexical NLP library, and a Ukrainian sentiment dictionary. It provides results in the form of sentiment scores for named entities at the sentence and text levels, as well as an overall sentiment evaluation of the analyzed content. The relevance of the research is determined by the growing need for accurate sentiment analysis in the context of large-scale digital information flows. Identifying emotional attitudes toward specific persons, organisations, or events has essential applications in monitoring public opinion, brand perception, political discourse, and financial market analysis. The scientific novelty lies in developing and implementing a method that supports Ukrainian-language texts and evaluates sentiment across negativity, neutrality, positivity, and emotionality dimensions. The practical significance is creating a software system capable of semantic sentiment analysis of textual content, achieving higher effectiveness than