A lightweight Secure Boot Mechanism for Protecting the Firmware of IOT Devices

Olena Danchenko[1,†], Inna Rozlomii[1,†], Andrii Yarmilko[2,*,†] and Serhii Naumenko[2,†]

[1] *Cherkasy State Technological University, 460 Shevchenko Blvd., Cherkasy, 18006, Ukraine*
[2] *Bohdan Khmelnytsky National University of Cherkasy, 81 Shevchenko Blvd., Cherkasy, 18031, Ukraine*

Abstract

The article presents the architecture of a lightweight Secure Boot mechanism specifically tailored for use in embedded IoT devices with limited hardware resources. Given constraints such as small memory capacity, lack of hardware cryptographic support, and the need to minimize power consumption, traditional secure boot schemes are often unsuitable for such systems. The proposed solution implements a multi-phase firmware integrity verification process prior to execution using the lightweight hash function SPONGENT-128/128/8, which offers acceptable resistance to attacks while maintaining low energy consumption and a compact implementation footprint. The mechanism verifies only critical memory regions, reducing system boot time to 21–29 ms and energy consumption to 10–11 μJ per check. A prototype implementation on STM32 and ESP8266 platforms confirmed the mechanism's effectiveness and accuracy: in all test scenarios, modified firmware was successfully detected, after which the devices transitioned into a protected mode. Additionally, the performance and memory footprint of various cryptographic primitives (SPONGENT, PHOTON, BLAKE2s) were evaluated, justifying the selection of specific algorithms based on the hardware configuration. The developed solution does not require a digital signature in its basic configuration but supports it in an extended version based on ECDSA with 160-bit curves, enabling source authentication even in the absence of a hardware trust module. The proposed mechanism can be integrated into practical solutions for medical, military, or industrial IoT systems, where balancing security, energy efficiency, and performance is critical. Promising directions for future research include obfuscation of the reference hash, implementation of trusted OTA updates, and expansion of support for emerging lightweight cryptographic algorithms.

Keywords

IoT, lightweight Secure Boot, embedded firmware, cryptographic hash, SPONGENT, energy efficiency

1. Introduction

The Internet of Things (IoT) is gradually transforming the concept of computing systems, shaping a new reality in which microcontrollers with limited hardware capabilities play a crucial role in mission-critical processes [1]. The embedded firmware of such devices serves as the core component that defines their functionality, interaction logic with the surrounding environment, and the overall reliability of the IoT system [2]. Compromising this layer of software opens the door to full control over the device and poses risks of unauthorized access, data manipulation, or loss of control.

The initial boot phase is particularly vulnerable, as the embedded firmware has not yet activated standard security mechanisms [3]. This moment presents an opportunity for the injection of foreign or modified code that may go undetected without proper authenticity verification. In fully

ICST-2025: Information Control Systems & Technologies, September 24-26, 2025, Odesa, Ukraine
* Corresponding author.
† These authors contributed equally.
elen_danchenko@ukr.net (O. Danchenko); inna-roz@ukr.net (I. Rozlomii); a-ja@ukr.net (A. Yarmilko); naumenko.serhii1122@vu.cdu.edu.ua (S. Naumenko)
0000-0001-5657-9144 (O. Danchenko); 0000-0001-5065-9004 (I. Rozlomii); 0000-0003-2062-2694 (A. Yarmilko); 0000-0002-6337-1605 (S. Naumenko)

featured computing systems, this problem is addressed by the Secure Boot mechanism, which performs cryptographic validation of the software before execution begins [4-6]. However, implementing a classical Secure Boot architecture in resource-constrained devices is often impractical due to its high demands on memory, energy consumption, and computational power.

As the number of autonomous IoT devices continues to grow and their security increasingly determines the resilience of entire digital infrastructures, there is a pressing need for a tailored, lightweight secure boot mechanism capable of operating in resource-limited environments [7].

The goal of this research is to develop and experimentally validate an efficient lightweight Secure Boot mechanism for protecting the embedded firmware of IoT devices against unauthorized modification and the execution of untrusted code.

2. Theoretical background and related works

Protecting embedded firmware in devices with limited computational resources is a highly relevant research area in the context of establishing trust in a device from the very beginning of its boot process [8–10]. One of the key approaches in this domain is the concept of Secure Boot, which involves verifying the authenticity and integrity of the software before handing off control to the next stage of the boot process [11]. In typical architectures, this is achieved through cryptographic verification of a digital signature or a hash generated using a public key provided by the manufacturer or system administrator [12].

In fully featured computing systems, Secure Boot mechanisms are widely supported at the UEFI, TPM, and other hardware-software platform levels [13]. These systems ensure a high level of control and enable the implementation of a trusted boot chain (chain of trust) [14]. However, for IoT devices and other microcontroller-based systems such as STM32, ESP8266, or RISC-V platforms, traditional Secure Boot schemes cannot be directly applied due to constraints in memory, energy consumption, and the absence of hardware cryptographic support.

Among existing approaches to adapting Secure Boot for such systems, the use of lightweight cryptographic algorithms-particularly hash functions based on BLAKE2s, SPONGENT, or PHOTON-stands out [15-17]. Lightweight implementations of digital signatures are also employed, such as those based on Ed25519 or ECDSA with shortened key lengths [18]. Research studies [19-21] have explored the possibility of using one-way hashing without signatures, significantly reducing the load on the microcontroller, though at the cost of limited source authentication. Other approaches, as seen in [22, 23], demonstrate the feasibility of phased booting with verification limited to critical code segments, achieving a balance between security and performance.

Additionally, the use of Secure Elements or Trusted Execution Environments (TEE) is being explored [24]. However, such solutions often exceed the capabilities of ultra-low-power devices performing only basic sensing or computational tasks. A persistent challenge is selecting a cryptographic core that ensures an adequate level of security without exceeding critical thresholds for energy consumption and boot latency [25]. This is especially crucial in medical and military IoT systems, where unpredictable delays or failures can lead to severe consequences.

Despite ongoing efforts to adapt secure boot mechanisms to resource-constrained environments, there remains a lack of generalized architectures that can seamlessly integrate Secure Boot into common IoT platforms without sacrificing efficiency. This highlights the need for a new approach to designing a lightweight Secure Boot mechanism based on compact cryptographic primitives, streamlined verification logic, and scalability across various computational configurations.

3. Technical requirements for lightweight Secure Boot

The operation of the Secure Boot mechanism in devices with low hardware resources requires a rethinking of standard secure boot requirements and their adaptation to the specific nature of embedded systems. Unlike fully featured computing platforms, IoT devices often lack advanced

security infrastructure, such as large non-volatile memory, dedicated hardware cryptographic modules, or secure key storage. In such environments, the development of a lightweight Secure Boot mechanism demands a clear definition of technical requirements that preserve the core properties of secure startup while remaining feasible under constrained conditions.

To better understand the technical limitations under which a lightweight Secure Boot must operate, it is essential to analyze the computational characteristics of common microcontroller platforms. Figure 1 presents a comparative overview of ROM, RAM, and CPU specifications for three representative Internet of Things architectures.

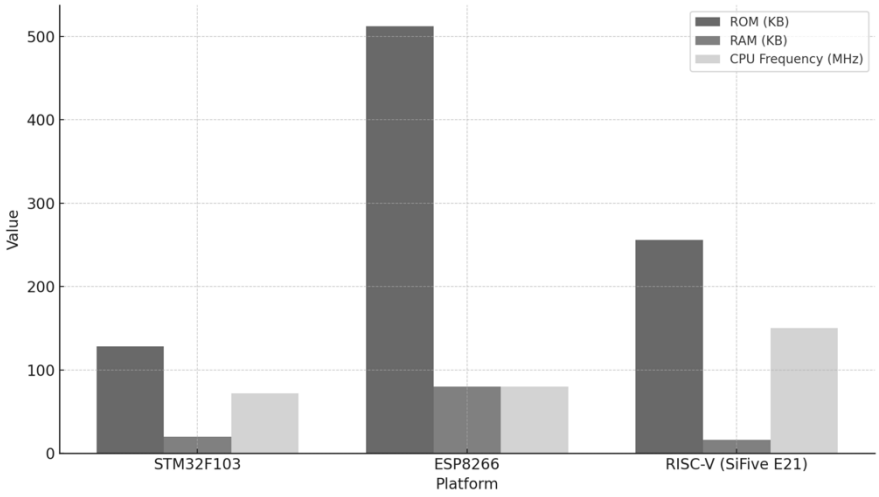

Figure 1: Resource constraints in typical IoT platforms [26].

As shown in Figure 1, even the most powerful microcontrollers in their class, such as the ESP8266, operate with limited RAM and computational capabilities, and none of the listed platforms provide built-in hardware cryptographic mechanisms. These limitations necessitate the use of lightweight cryptographic primitives and minimal verification logic during the secure boot process.

First and foremost, a critical requirement is ensuring the integrity verification of the embedded firmware before it is executed. This involves verifying a checksum or hash of the main firmware against a predefined reference value. To achieve this, a lightweight cryptographic hash function must be used one that offers sufficient resistance to collisions and forgery while maintaining low energy consumption and a compact implementation footprint. Functions such as SPONGENT, PHOTON, or simplified versions of BLAKE2s can be utilized on compatible microcontroller architectures.

When selecting a hash function under resource-constrained conditions, it is important to consider not only cryptographic strength but also the technical characteristics of the implementation. Table 1 provides a comparison of lightweight hash functions that may be used in the Secure Boot mechanism on STM32 and ESP8266 microcontrollers.

As shown in Table 1, SPONGENT has the lowest energy consumption but is less resistant to collisions compared to PHOTON and BLAKE2s. Meanwhile, BLAKE2s offers the highest level of cryptographic robustness but requires more computational resources.

The second critical requirement is protecting the bootloader itself the component responsible for verifying the main firmware.

This element should be stored in a secure memory region, preferably in non-volatile or read-only memory. In the absence of hardware-based memory partitioning, it is important to ensure at least a unified trust chain verification mechanism, based on cryptographic linkage between the bootloader and the main firmware.

Table 1
Comparison of Lightweight Hash Functions for Secure Boot Applications

Hash Function	Implementa -tion Size (bytes/gates)	Computation Time	Collision Resistance	Power Consumption	Support on STM32 / ESP8266
SPONGENT	2260 gates	Low	Moderate	Very Low	Yes / Partial
PHOTON	2177 gates	Medium	High	Low	Yes / Experimental
BLAKE2s	≈ 1300 bytes	High	Very High	Medium	Yes / Optimized

Another key parameter is the available memory for implementing Secure Boot—both code and runtime memory. On many common IoT platforms, flash memory may be limited to 128 or 256 KB, with RAM availability restricted to just a few kilobytes. Therefore, any algorithm or verification structure must be implemented compactly, with minimal overhead, and must not interfere with the device's primary functionality.

Execution time and main program startup latency must also be considered. A lightweight Secure Boot mechanism should ensure minimal initial verification time, as even small delays can be critical in many real-time or sensor-based devices. The balance between security level and boot speed should be determined experimentally, based on acceptable timing thresholds for the specific application.

Additionally, a reliable source of the reference hash or digital signature must be available and protected from tampering. In the absence of hardware key storage, such as a Trusted Platform Module or Secure Element, it is advisable to store the reference value in encrypted form or embed it directly into the initial bootloader.

Figure 2 illustrates an extended architectural model of the Secure Boot mechanism for resource-constrained microcontrollers. This model includes not only basic hash comparison but also secure storage, random number generation, and a public key reference, reflecting practical implementations in real-world IoT devices.

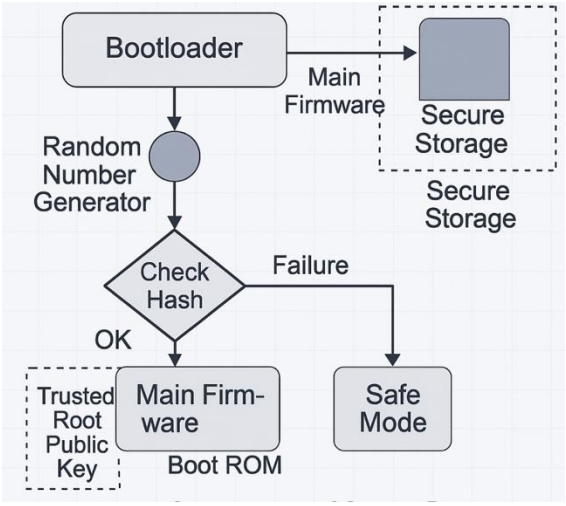

Figure 2: Minimal components of Secure Boot for a microcontroller.

The inclusion of secure storage for hashes and trusted keys enables authenticity verification without relying on heavy cryptographic modules, while a fallback path ensures resilience in case of tampering or hash mismatches.

A lightweight Secure Boot mechanism must meet a set of requirements that provide basic cryptographic verification of software authenticity while maintaining system operability within constrained computational, energy, and timing resources.

4. The proposed architecture of the lightweight Secure Boot mechanism

In the proposed approach, Secure Boot is implemented as a lightweight multi-phase verification of the integrity and authenticity of embedded firmware, tailored for IoT devices with limited computational resources. The architecture takes into account constraints such as minimal memory usage, limited processing power, and system responsiveness, while providing a basic level of security without the need for hardware cryptographic modules.

4.1. Overall operation scheme

The Secure Boot mechanism in the proposed architecture is implemented as a sequence of integrity verification stages for the main firmware prior to its execution. Its purpose is to ensure that only authentic and unmodified software is allowed to run. Immediately after power-on or device reset, the system initializes the bootloader, which serves as the sole entry point into the system before the main program is executed.

The bootloader accesses a secure memory region where the reference hash of the main firmware is stored. Simultaneously, it computes the hash of the current firmware content using a lightweight hash function. The system then compares the reference hash with the computed hash. If they match, control is handed over to the main firmware. If not, the device enters a lock state (Safe Mode) or switches to a fallback scenario.

Figure 3 illustrates the overall Secure Boot logic, highlighting key components and processing flows. Upon successful verification, the system proceeds to execute the main functional code.

As shown in Figure 3, the proposed structure implements a multi-phase approach in which key verification steps are performed before the main code is executed. By relying solely on lightweight cryptographic primitives and verifying only critical memory regions, such as the interrupt vector table, initialization blocks, or data handling functions, the system minimizes computational overhead. Additional components, such as the random number generator, public key, and secure storage, are activated only when needed and are not involved in the main verification loop, which reduces energy consumption.

This approach allows for flexible scalability of the architecture according to the capabilities of a specific hardware platform while maintaining basic authenticity and integrity checks. As a result, it achieves an effective balance between boot-time performance, energy efficiency, and compatibility with the limited resources of typical IoT devices.

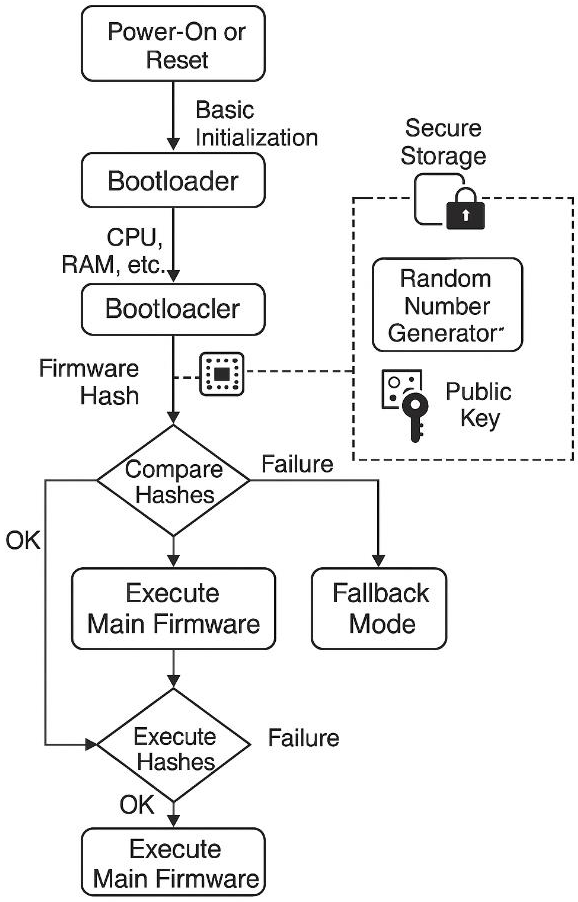

Figure 3: General workflow of the proposed lightweight Secure Boot mechanism.

4.2. Cryptographic elements used

Given the limited computational capabilities of the target devices, the SPONGENT-128/128/8 hash function was selected for its balance between attack resistance and compact implementation. This function is based on a sponge construction composed of permutation rounds and bitwise operations, allowing the generation of variable-length hash outputs from limited input blocks. A notable feature of SPONGENT is its ability to operate efficiently even on microcontrollers with clock speeds below 100 MHz, consuming minimal energy (less than 10 μJ per operation). The hashing algorithm can be represented as (1).

$$Z = Sponge[f, pad, r](M), \tag{1}$$

where f – s the internal permutation function (in SPONGENT, built from multiple rounds), pad – is the padding algorithm used to extend the message to the required length $r = 8$ – is the number of bits processed per round, M – is the input message (firmware content or its critical segments), Z – is the resulting hash output.

The processing sequence is as follows:

1. Initialization of the internal state – $S_0 = IV$.
2. Absorbing rounds – $S_{i+1} = f(S_i \oplus m_i)$ for each input block m_i.
3. Output generation: extract the first n bits from the final state $Z = first_n(S_k)$.

This approach enables the generation of variable-length hashes without the need for complex or energy-intensive operations. In critical real-time devices, SPONGENT allows integrity verification with minimal system startup delay.

In cases where the device architecture supports the use of more advanced algorithms, BLAKE2s may be employed, offering higher cryptographic robustness and well-established implementations in various security libraries. PHOTON may also be used, as it demonstrates strong resistance to differential and linear cryptanalysis while maintaining a compact implementation suitable for platforms with memory constraints as low as 8 KB.

In the basic version of the architecture, the Secure Boot mechanism does not use a digital signature, which helps reduce code size, the number of large-number operations, and startup delay. However, in scenarios where firmware source authentication is critical, an extended version is provided that utilizes a lightweight digital signature based on ECDSA with 160-bit curves (e.g., secp160r1). This type of signature requires a minimum of two elliptic curve multiplications during verification but can still ensure authenticity even if an attacker has access to external memory.

The signature verification procedure in this case is carried out as follows:

1. The message hash (firmware) is computed – $e = HASH(M)$.
2. Auxiliary values are then determined – $\varpi = s^{-1} \, mod \, n \;\; u_1 = e \cdot \varpi \, mod \, n, u_2 = r \cdot \varpi \, mod \, n.$
3. A point on the curve is calculated – $P = u_1 G + u_2 QP$, where G – is the base point, Q – is the public key, and r, s_r — are the signature parameters.
4. Authenticity check – $r \equiv x_P \, mod \, n.$

Where x_P – is the x-coordinate of point P, resulting from the computation. All elliptic curve operations are implemented using arithmetic over a prime field ppp, and the public part of the key may be stored in ROM, embedded in the bootloader, or securely extracted from an external source.

The authenticity of the hash is verified using a public key, which may be stored in one of the following ways:

- in non-volatile ROM memory (secured option),
- hardcoded into the initial bootloader,
- loaded from an encrypted source, assuming a decryption module is available.

This approach enables the establishment of a one-way chain of trust even in the absence of a full-fledged TPM or Secure Element. Digital signature generation or verification is activated only during the update phase or when verifying the reference hash, avoiding impact on the main boot cycle and helping to prevent energy spikes during critical operation modes.

4.3. Code storage and verification

The reference hash of the main firmware is stored in a protected flash memory region, to which the bootloader has read-only access. In some implementations where hardware-based memory partitioning is available, a dedicated ROM segment is used to ensure the immutability of the reference value.

A key feature of the implementation is that it does not verify the entire firmware but only its most critical sections, such as the initialization code, interrupt vector table, and network traffic handling functions. This reduces the computational load and allows verification to be completed within an acceptable time frame. The boot strategy is based on a fail-stop logic: if a mismatch is detected, execution of the main program is denied, and the device enters either a locked or fallback mode.

The sequence of steps for verifying the authenticity and integrity of the software includes the following stages:

1. After power-up or device reset, the bootloader is activated, performing basic system initialization and memory access verification.

2. The boundaries of critical firmware regions to be verified are defined. These may include areas containing initialization code, the interrupt vector table, or functions responsible for handling network traffic.
3. The bootloader initiates hashing of the selected segments using a lightweight hash function (SPONGENT, PHOTON, or BLAKE2s), adapted for low-performance environments with limited energy consumption.
4. The computed hash is compared to the reference value stored in a protected memory region. The comparison is performed without storing intermediate copies in volatile memory, preventing tampering during the verification process.
5. If the hashes match, the bootloader transfers control to the main program, which begins execution according to the loaded logic.
6. If a mismatch between the computed and reference hash is detected, the system immediately halts the boot process and enters a secure state, such as Safe Mode, interface lockout, or initiates a recovery or update process from a trusted source.
7. To enhance reliability, some implementations allow storing multiple valid hashes, enabling support for several firmware versions and implementing a fallback boot mechanism.

The proposed verification sequence adheres to security principles while accounting for the platform's technical constraints and preserves system performance even under critically limited resources.

5. Experimental evaluation and performance assessment

To evaluate the functionality of the proposed Secure Boot architecture, an experimental implementation was carried out on two typical IoT-class platforms: the STM32F103C8T6 (an ARM Cortex-M3-based microcontroller with a 72 MHz clock) and the ESP8266 NodeMCU (Tensilica L106 processor, 80 MHz). Both microcontrollers are widely used in low-power projects and feature limited RAM, making them suitable candidates for assessing the mechanism's efficiency under resource-constrained conditions.

A Secure Boot prototype was implemented, which included a 5.1 KB bootloader, a SPONGENT-128/128/8 hash computation module, secure storage of the reference hash, and a verification routine executed before launching the main application. A 32 KB firmware program was used for testing, with intentional modifications made to critical sections such as the interrupt vector table and data processing functions.

Programming was done in C using the GCC compiler (for STM32) and Arduino Core (for ESP8266). Timing, energy consumption, and code size were measured using STM32CubeMonitor, an INA219 sensor (for real-time current measurement), and a Tektronix oscilloscope. Experimental results showed that the full verification cycle took 21.7 ms on the STM32 and 29.1 ms on the ESP8266. SPONGENT hash computation accounted for 18.2 ms and 25.4 ms, respectively, with energy consumption per verification not exceeding 10–11 microjoules. The compiled Secure Boot code size was 6.4 KB for STM32 and 7.1 KB for ESP8266.

Three key metrics were used to assess efficiency: verification time, energy consumption during startup, and the bootloader's code size. Test scenarios included both verifying unmodified firmware and simulating tampered code. The experimental results are presented in Table 2.

As shown in the table, both platforms demonstrate high accuracy in detecting modified firmware without introducing noticeable system startup delays. The lowest values for energy consumption and verification time were achieved through the use of lightweight cryptographic primitives and limiting verification to only the most critical memory segments. This makes it possible to effectively integrate protection into systems that are sensitive to latency or rely on battery-powered operation. The results confirm the viability of the proposed architecture as a flexible solution for secure boot in resource-constrained devices.

Table 2
Experimental Evaluation of the Proposed Secure Boot Implementation

Metric	STM32F103C8T6	ESP8266 NodeMCU
Bootloader size	6.4 KB	7.1 KB
Hashing time (SPONGENT)	18.2 ms	25.4 ms
Full verification time	21.7 ms	29.1 ms
Energy per verification	9.8 μJ	11.3 μJ
Memory overhead (RAM)	< 512 bytes	< 768 bytes
Detection of tampering	100%	100%
Main firmware protection reaction	Safe Mode or rollback	Safe Mode or rollback

The system successfully detected firmware tampering: in cases where control bytes were altered, the device entered a safe mode without executing the main application. This demonstrates the mechanism's ability to resist unauthorized modifications even in scenarios where physical access to the device's memory is possible. Compared to traditional approaches that rely on digital signatures and elliptic curve cryptography, the proposed lightweight hashing variant achieved 3–4 times faster startup and an order of magnitude lower energy consumption.

However, the basic implementation does not address protection of the reference hash against direct reading, which could be critical in cases of physical compromise. Future research is planned to focus on integrating obfuscation techniques, the use of a Secure Element, and implementing trusted remote (OTA) updates with server-side cryptographic validation. The results obtained confirm the feasibility of deploying the mechanism in practical IoT solutions, where the combination of security and energy efficiency is essential.

6. Discussion

The experimental results confirmed that the proposed lightweight Secure Boot mechanism can be effectively implemented on resource-constrained microcontrollers, maintaining a balance between performance and a basic level of security. The use of a lightweight hash function and the restriction of verification to critical firmware segments enabled acceptable boot times and minimal energy consumption, which are particularly important for autonomous IoT devices. The successful detection of modified firmware in all test scenarios demonstrates the architecture's ability to counter basic threats without requiring complex hardware modules.

However, certain limitations remain relevant. Specifically, storing the reference hash in plaintext even within protected flash memory leaves room for potential attacks in cases of physical access to the device. Additionally, the current implementation lacks a secure firmware update mechanism, which complicates deployment in dynamic environments with frequent updates. Integration with external trust sources or remote verification servers is also not addressed, which is essential for establishing a full chain of trust.

Promising directions for future research include integrating encryption or obfuscation mechanisms for the reference hash, implementing an extended version with digital signature support, and using external hardware components such as a Secure Element or TPM. Special attention should be given to optimizing energy consumption during updates and developing an architecture that supports secure remote updates with server-side verification. It is also advisable to expand support for modern cryptographic algorithms optimized for energy efficiency and to include behavior monitoring tools during boot to build an adaptive, self-healing security system.

7. Conclusions

This article presents the architecture of a lightweight Secure Boot mechanism adapted to the resource-constrained conditions of IoT devices. The proposed approach is based on the use of lightweight cryptographic primitives, particularly the SPONGENT-128/128/8 hash function, and

focuses on verifying only the critical segments of the firmware to minimize energy consumption and startup time. The solution does not require hardware cryptographic modules and can be implemented on popular microcontrollers such as STM32 and ESP8266.

The experimental implementation confirmed the feasibility and efficiency of the developed mechanism: the verification delay was under 30 ms, and energy consumption remained within 10–11 microjoules. All firmware modifications were successfully detected, demonstrating the suitability of the proposed approach for systems where both security and energy efficiency are critical.

The results of this study open up possibilities for further improvement of the architecture, including the integration of digital signature support in the extended configuration, obfuscation of reference hashes, and incorporation of a secure software update module. The proposed solution is flexible and scalable across different device classes, making it a potential foundation for building trusted IoT environments.

Acknowledgements

This research was funded by the Ministry of Education and Science of Ukraine under grant 0125U000637.

Declaration on Generative AI

Generative artificial intelligence tools were used exclusively for creating the diagram presented in Figure 1.

References

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

[2] A. Yarmilko, I. Rozlomii, S. Naumenko, 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", Springer Nature Switzerland, Cham, 2024, pp. 235-249.

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

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

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

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

[7] I. Rozlomii, A. Yarmilko, S. Naumenko, P. Mykhailovskyi, 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, 2024, pp. 99-109.

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

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

[10] M. Grisafi, M. Ammar, M. Roveri, B. Crispo, {PISTIS}: Trusted computing architecture for low-end embedded systems, in: 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 3843-3860.
[11] S. Arunkumar, 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), IEEE, 2024, pp. 1-4.
[12] T. Jyothi, S. Jain, Tpm based secure boot in embedded systems, in: 2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC), IEEE, 2023, pp. 786-790.
[13] S. Schmidt, M. Tausig, M. Koschuch, B. Sheng, M. Hudler, P. Puddu, C. Konstantinou, A. Oluwaferanmi, Leveraging Trusted Platform Modules (TPM) for Cryptographic Anchoring and Remote Attestation of UEFI Capsule Updates in Secure Boot Environments, 2025. URL: https://www.researchgate.net/publication/392693928_Leveraging_Trusted_Platform_Modules_TPM_for_Cryptographic_Anchoring_and_Remote_Attestation_of_UEFI_Capsule_Updates_in_Secure_Boot_Environments.
[14] Z. Ling, H. Yan, X. Shao, J. Luo, Y. Xu, B. Pearson, X. Fu, Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes, Journal of Systems Architecture 119 2021 102240.
[15] P. Tandel, J. Nasriwala, Efficient authentication framework with Blake2s and a hash-based signature scheme for Industry 4.0 applications, International Journal of Intelligent Engineering Informatics 12 4 (2024) 410-432.
[16] S. B. Hiremath, S. Heblikar, M. S. Javali, M. Tejas, N. C. Iyer, Side Channel Analysis and Hardware Implementation of AES CBC Algorithm on Artix A7 FPGA Development Boards, in: International Conference on ICT for Sustainable Development, Springer Nature Singapore, Singapore, 2024, pp. 111-120.
[17] M. Al-Shatari, F. A. Hussin, A. A. Aziz, T. A. E. Eisa, X. T. Tran, IoT Edge Device Security: An Efficient Lightweight Authenticated Encryption Scheme Based on LED and PHOTON, Applied Sciences 13 18 (2023) 10345.
[18] J. Brendel, C. Cremers, D. Jackson, M. Zhao, The provable security of ed25519: theory and practice, in: 2021 IEEE Symposium on Security and Privacy (SP), IEEE, 2021, pp. 1659-1676.
[19] R. Román, R. Arjona, I. Baturone, A lightweight remote attestation using PUFs and hash-based signatures for low-end IoT devices, Future Generation Computer Systems 148 (2023) 425-435.
[20] H. A. N. Songshen, X. U. Kaiyong, Z. H. U. Zhiqiang, G. U. O. Songhui, L. I. U. Haidong, L. I. Zuohui, Hash-based signature for flexibility authentication of IoT devices, Wuhan University Journal of Natural Sciences 27 1 (2022) 1-10.
[21] A. Chakraborty, A. Biswas, 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), IEEE, 2025, pp. 1309-1316.
[22] B. Li, W. Dong, Edge-centric programming for iot applications with automatic code partitioning, IEEE Transactions on Computers 71(10) (2021) 2408-2422.
[23] Y. Dong, S. Duan, F. Lyu, P. Zhao, Y. Zhang, J. Ren, Y. Zhang, NC-Load: On-Demand Program Loading and Running for Computing Sharing Among IoT Devices, IEEE Internet of Things Journal 12(9) (2025) 12242-12256. doi: 10.1109/JIOT.2024.3520584.
[24] A. Muñoz, R. Ríos, R. Román, J. López, A survey on the (in) security of trusted execution environments, Computers & Security 129 (2023) 103180.
[25] I. Rozlomii, S. Naumenko, P. Mykhailovskyi, V. Monarkh, Resource-Saving Cryptography for Microcontrollers in Biomedical Devices, in: 2024 IEEE 5th KhPI Week on Advanced Technology (KhPIWeek), IEEE, 2024, pp. 1-5.
[26] V. A. Thakor, M. A. Razzaque, M. R. Khandaker, Lightweight cryptography algorithms for resource-constrained IoT devices: A review, comparison and research opportunities, IEEE Access 9 (2021) 28177-28193.