The method for verifying firmware integrity in IoT devices for secure boot using lightweight hash functions*

Inna Rozlomii[1,*,†], Emil Faure[1,2,†], Andrii Yarmilko[3,†] and Serhii Naumenko[3,†]

[1] *Cherkasy State Technological University, 460 Shevchenko ave., 18006 Cherkasy, Ukraine*
[2] *State Scientific and Research Institute of Cybersecurity Technologies and Information Protection, 3/6 M. Zaliznyaka str., 03142 Kyiv, Ukraine*
[3] *Bohdan Khmelnytsky National University of Cherkasy, 81 Shevchenko ave., 18031 Cherkasy, Ukraine*

Abstract
The paper is devoted to the development of a method for verifying the integrity of firmware for IoT devices, focused on the conditions of limited computing resources and minimal power consumption. The method is based on the use of lightweight hash functions, such as SPONGENT, PHOTON, QUARK and LESAMNTA-LW, which provide computational efficiency on low-end microcontrollers. A multi-level approach is proposed, in which firmware segments are evaluated by weight coefficients depending on their criticality for security, and the aggregated control value is formed taking into account the importance of each segment. To increase protection against replay attacks, session markers are integrated into hashing, which add context dependency. The adaptive nature of the check allows you to dynamically change the depth of analysis depending on the state of the device—for example, the battery charge level or processor load. The experimental part of the work covers testing the method on popular microcontrollers STM32F072, ESP8266 and ATmega328P. The study included an assessment of verification time, memory consumption, power consumption, and resistance to firmware modification attacks and reuse of previous values attacks. Special attention is paid to partial verification scenarios, which are relevant for devices with limited resources. The proposed method is considered suitable for a wide range of IoT applications, including autonomous sensors, energy modules, medical devices, and transportation systems. The results demonstrate the possibility of effective secure boot even on platforms without hardware cryptography support.

Keywords
limited resources, IoT, lightweight hashing, session tokens, weighted aggregation, STM32F072, ESP8266, ATmega328P

1. Introduction

The Internet of Things (IoT) today encompasses a vast array of devices, from consumer electronics to medical implants and industrial sensors [1–6]. As the number of IoT devices increases, so does the level of threats to them, especially when it comes to firmware modification, which is one of the key targets of attacks [7, 8]. Firmware modification or replacement can lead to loss of functionality, leakage of confidential data, or complete device takeover [9]. Ensuring firmware integrity is becoming a prerequisite for Secure Boot, but in practice its implementation is complicated by the limited resources of IoT devices, in particular limitations in terms of memory, processing power, and power consumption [10, 11].

Traditional cryptographic hash functions, such as SHA-2 or SHA-3, demonstrate high resistance to attacks, but their use in resource-constrained environments is of little use [12]. They consume a significant amount of RAM, require a powerful processor, and increase the overall boot time [13–15]. These limitations have encouraged research groups to develop lightweight hash functions aimed at IoT needs, including SPONGENT, PHOTON, QUARK, and LESAMNTA-LW [16–18]. They

CSDP'2025: Cyber Security and Data Protection, July 31, 2025, Lviv, Ukraine
* *Corresponding author.*
† These authors contributed equally.

inna-roz@ukr.net (I. Rozlomii); e.faure@chdtu.edu.ua (E. Faure); a-ja@ukr.net (A. Yarmilko); sergey.naumenko94@vu.cdu.edu.ua (S. Naumenko)
0000-0001-5065-9004 (I. Rozlomii); 0000-0002-2046-481X (E. Faure); 0000-0003-2062-2694 (A. Yarmilko); 0000-0002-6337-1605 (S. Naumenko)

allow for significant reductions in computational costs and power consumption while maintaining an acceptable level of resistance to cryptanalytic attacks.

Against the background of these limitations, there is considerable scientific interest in analyzing the possibility of using lightweight hash functions for integrity verification, as they are specifically designed for resource-constrained environments. Their application opens up new prospects for embedded systems, allowing for secure booting even where the use of classical algorithms is impractical or technically impossible.

Figure 1 presents a general diagram of a secure boot of an IoT device using a lightweight hash function, showing the main stages—storing a check hash, calculating the hash of the loaded firmware, and comparing the values before launching the main program.

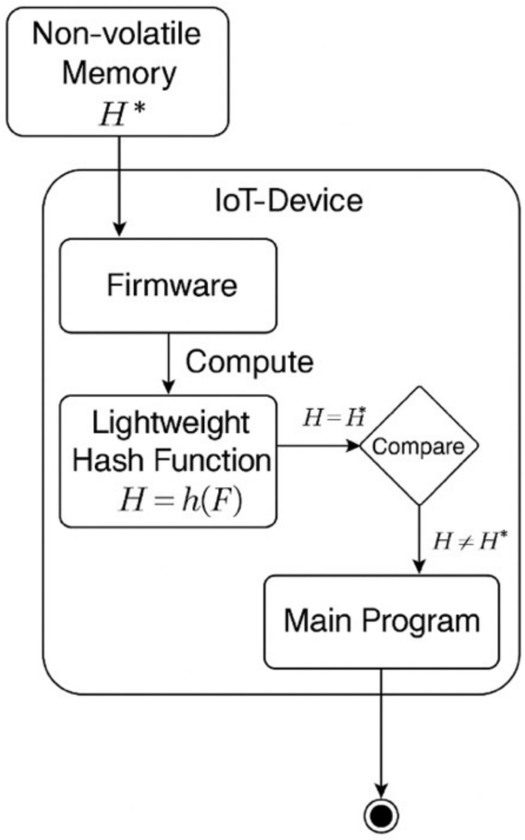


Figure 1: General scheme of verifying the integrity of the firmware of an IoT device using a lightweight hash function

The scheme clearly shows the sequence of actions: first, a reference hash value is read from non-volatile memory, then a hash of the current firmware version is calculated, after which both values are compared. In case of a match, the system proceeds to the next stage of loading, otherwise it blocks the loading or activates recovery procedures.

The specified scheme illustrates the key stages of implementing Secure Boot in an IoT device, emphasizing the role of the hash function as the central element responsible for verifying the authenticity of the loaded code. Importantly, this approach allows integrating the verification mechanism at the software level without the need for hardware crypto modules, while maintaining the limited amount of memory and low power consumption that are critical for autonomous sensor nodes, medical implants and other similar systems.

The aim of the research is to develop a method for verifying firmware integrity for IoT devices, based on the use of lightweight hash functions and taking into account the specifics of resource-constrained environments, in particular, minimizing computational costs, memory, and power consumption while ensuring an appropriate level of cryptographic stability.

2. Related works

Providing Secure Boot in IoT devices has attracted considerable attention from researchers in recent years [19–21]. The main goal of Secure Boot is to ensure that only verified and authentic code is executed by the device during startup. According to [22], the principle of operation of Secure Boot is based on cryptographic verification of digital signatures or hash values of program code stored in the trusted memory of the device. Works [23, 24] demonstrate the implementation of such approaches based on microcontrollers with hardware support for cryptography, for example, Trusted Platform Module (TPM) or ARM TrustZone, however, these solutions are not suitable for low-end microcontrollers due to their high cost and power consumption.

Current methods for ensuring integrity in microcontrollers are mostly focused on a compromise between the level of security, computational costs and resource consumption. For example, [25] describes a software implementation of Secure Boot for STM32 microcontrollers, where the SHA-256 hash function is used to verify the integrity of the firmware. The results of the study showed that the use of SHA-256 allows for high cryptographic stability, but leads to an increase in the device boot time by 30–50% depending on the firmware size, and also requires a significant amount of RAM (several tens of kilobytes), which is critical for microcontrollers with limited RAM. The article [26] considers approaches to optimizing such solutions, including partial hashing, when only the most critical code segments are checked, or a phased integrity check per segment, which allows distributing the load on computing resources. However, the authors emphasize that such simplifications potentially create new attack vectors, for example, selective substitution of uncontrolled segments or attacking actions during intermediate checks.

The development of lightweight hash functions has become a separate research area, since they are specifically designed for use in embedded and sensor systems [27]. In [28], a classification of such functions by design approaches was carried out: sponge-constructions, Davies–Meyer block schemes and double-block Hirose. SPONGENT [29], which belongs to sponge-constructions, demonstrates a noticeable reduction in computational costs compared to SHA-2 due to the use of a compact state block and simple bitwise operations XOR, AND, ROT, which allows minimizing memory consumption to the level of 1–2 kB. PHOTON [30], built on the principle of Substitution–Permutation Network (SPN), combines a small hardware implementation area and resistance to the main types of attacks, such as differential and linear cryptanalysis, which is confirmed by numerous studies on ARM Cortex-M and AVR microcontrollers. QUARK [31], designed as a serialized stream cipher-like design, is particularly effective in scenarios with tight energy constraints, as it requires a minimum number of clock cycles per byte of data. LESAMNTA-LW [32] is positioned as a solution for ARM Cortex-M platforms, providing a balance between speed (due to extensive modular addition and permutation operations) and resistance to collisions and preimage attacks, while maintaining a small code size and low memory consumption.

Despite the presence of a significant amount of research, there are open problems that need to be solved. These include choosing the optimal hash function for specific IoT scenarios, mathematical modeling of the security-resources ratio, adapting Secure Boot for microcontrollers without hardware cryptography support, and providing protection against attacks at the physical level (e.g., fault injection). Recent advances in access control mechanisms emphasize the importance of integrating policy-as-code frameworks to enforce role-based and attribute-based access control, thereby enhancing the security of IoT environments [33]. Additionally, improvements in device identification and authentication accuracy through electromagnetic (EM) measurements provide promising avenues for strengthening hardware-level trustworthiness in constrained IoT devices [34]. Furthermore, the design of combined pseudo-random sequence generators, as well as generators based on mathematical constants such as ln 2, contribute to the development of lightweight and secure cryptographic primitives suitable for IoT firmware integrity verification [35, 36]. These approaches collectively support the enhancement of secure boot processes and firmware integrity validation under resource constraints. Further research should focus on finding methods that consider the balance between cryptographic robustness, hardware platform limitations, and practical time and power consumption requirements.

3. Method for verifying firmware integrity in IoT devices

The proposed method for verifying the integrity of firmware in IoT devices is based on the concept of multi-level, weighted and context-dependent verification, which allows taking into account the heterogeneity of the importance of individual sections of the code, the hardware limitations of the device and the current mode of its operation. A feature of the method is that for each code segment, weight coefficients are determined that reflect its criticality for system security. The aggregated control value is formed on the basis of calculated local hashes taking into account these weights, which allows increasing the depth of verification for the most vulnerable components without a significant increase in the load on the system.

Unlike classic Secure Boot schemes, where a single hash value of the entire firmware is compared, the proposed method uses lightweight hash functions, for example, SPONGENT, PHOTON, QUARK or LESAMNTA-LW, which provide a balance between cryptographic stability and resource efficiency. This opens up the possibility of implementing integrity checking even on low-power microcontrollers, such as STM32F0, ESP8266, AVR, which have a limited amount of RAM (up to several kilobytes) and computing resources. The method takes into account not only memory limitations, but also minimizing power consumption, which is especially relevant for autonomous sensors, medical implants, IoT modules in power grids and transport systems.

A key element of the approach is the adaptive selection of the depth of the check depending on the state of the device, for example, the battery charge level, temperature regime or processor load. In scenarios with limited energy resources, a partial check mode is activated, where only the most critical segments are checked, while under normal conditions a full check is performed. This approach allows the IoT device to dynamically balance between protection and autonomy, maintaining the appropriate level of security in conditions of limited resources.

Additionally, the method involves the implementation of context markers (e.g., time tokens or session identifiers) that are integrated into the hashing process. This increases the system's resistance to replay attacks, since the verification is carried out not only on the basis of the code, but also taking into account the session context, which significantly complicates the preparation of fake firmware. The proposed approach combines the concepts of cryptographic robustness, resource efficiency, and adaptability, which allows it to be scaled for a wide range of IoT applications.

The algorithm of the proposed method for verifying the integrity of the firmware includes multi-level hashing with weighted aggregation, adaptive selection of the verification depth, and the use of context markers to protect against replay attacks. Let the firmware F be divided into segments $F=\{f_1, f_2, \ldots, f_n\}$, each of which is assigned a weight coefficient w_i, reflecting its criticality, with $\sum_{i=1}^{n} w_i = 1$.

The choice of an aggregation model based on weight coefficients is associated with the need to take into account the criticality of functional modules. The coefficients w_i are formed based on a preliminary risk analysis, which takes into account the role of the segment in system security, the frequency of access to resources and the probability of attacks. This approach allows you to achieve a flexible balance between security and resource costs.

To form the aggregated control value, the weighted average aggregation model (1) is used.

$$H_{agg}=\frac{\sum_{i=1}^{n} w_i \cdot H_i}{\sum_{i=1}^{n} w_i}, \tag{1}$$

where H_i is the local hash value of the i-th segment, w_i is its weighting factor, n is the number of segments. This scheme allows to increase the contribution of critical components to the final result, while reducing the influence of secondary modules. Alternatively, for devices with very tight constraints, it is possible to use a simple additive model (2).

$$H_{agg}=\sum_{i\in S} H_i, \tag{2}$$

where $S \subseteq \{1,\ldots,n\}$ is a subset of the most important segments selected by the criterion w_i exceeds the set threshold value. The aggregation model is selected at the system configuration stage in accordance with the target usage scenario.

At the reference profile formation stage, the following is performed:

1. Calculation of local hashes $H(f_i)$ for each segment using a lightweight hash function $H(x)$, such as SPONGENT or PHOTON.
2. Formation of an aggregated hash value—$H_{agg}=\sum_{i=1}^{n} w_i \cdot H(f_i)$.
3. Writing H_{agg} to trusted non-volatile memory together with a time or session token T used as a salt value.

During device boot, the following is performed:

1. Reading H_{agg}^{c} and T^{e} from trusted memory.
2. Depending on the state of the device (e.g., battery level), determine the set $C(F) \subseteq F$ to be tested.
3. Calculation—$H_{agg}^{c}=\sum_{j\in C(F)} w_j \cdot H(f_j \vee T^{e})$, where $\vee$ is a concatenation operation that binds hashing to the session token.
4. Compare H_{agg}^{c} and H_{agg}^{e} and make a decision (3).

$$D\left(H_{agg}^{c}, H_{agg}^{e}\right)=\begin{cases}1, \mathit{if}\; H_{agg}^{c}=H_{agg}^{e} \\ 0, \mathit{if}\; H_{agg}^{c}\neq H_{agg}^{e}\end{cases}. \tag{3}$$

The computational complexity of the algorithm for full verification is estimated as $O(n \cdot C_H)$, where n is the number of segments, C_H is the average complexity of calculating the hash for one segment. For partial verification, which is activated when resources are reduced, the complexity decreases in proportion to the number of verified segments $C(F) \vee$. This allows reducing the verification time by 30–60% depending on the selected configuration.

Particular attention is paid to the selection of optimal w_i—for example, larger values are assigned to areas responsible for communications, hardware control, or secure data storage, while auxiliary modules (e.g., user interfaces) are given minimal weight. The method is focused on checking only static segments, i.e. parts of the code that do not change during execution. Dynamic modules that are loaded during operation are not covered by the current model and require separate solutions, such as digital signature verification or runtime integrity monitoring.

Figure 2 presents a flowchart of the algorithm, which demonstrates the process of multi-level validation, adaptive segment selection, and the use of session tokens. This algorithm allows for high scalability—it is suitable for both simple sensors and more powerful edge devices, where advanced cryptographic operations are available. The introduction of weighting factors and context markers increases resistance to attacks even in scenarios with limited energy and computing power.

If a mismatch in control values is detected, the device enters a protected mode, which involves blocking the launch of the main code, recording the event in the system log and, if possible, transmitting a message to an external monitoring system. For some categories of devices, for example, in medical applications, automatic activation of the firmware recovery mode with restoration of a previously saved valid firmware version is provided.

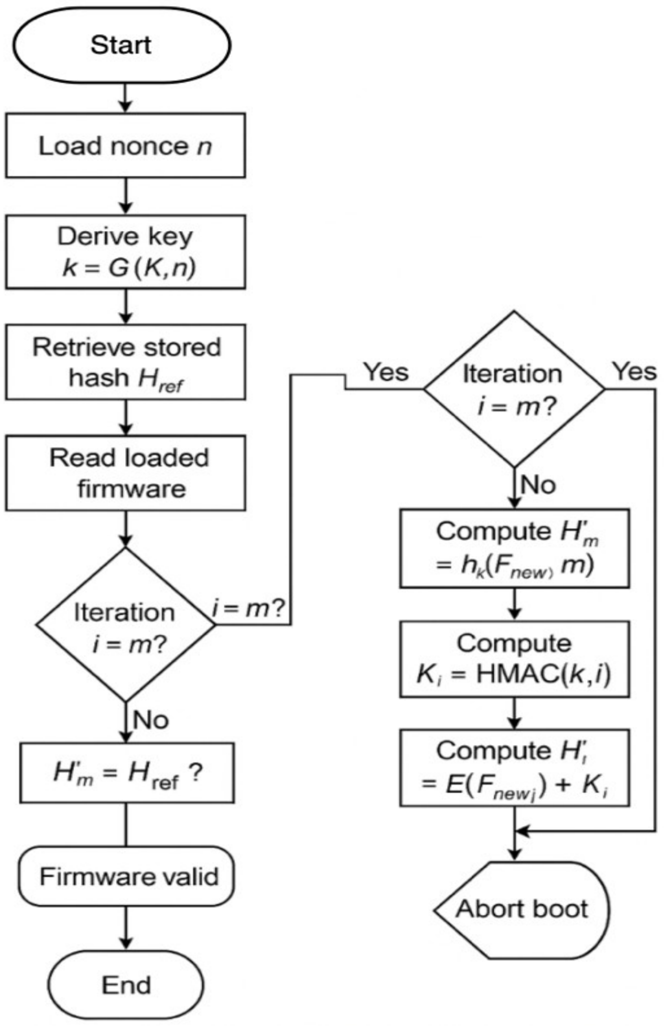


Figure 2: Block diagram of the firmware integrity verification algorithm with weighted aggregation and adaptive checking

4. Experimental evaluation of the effectiveness of the method

4.1. Experimental conditions

The experimental evaluation of the method's effectiveness was carried out on hardware platforms representing typical classes of resource-constrained IoT devices. Three microcontrollers were used for the study:

- STM32F072 (32-bit, Cortex-M0, 48 MHz frequency, 64 KB Flash, 16 KB SRAM, 12-bit ADC, USB, SPI, I2C);
- ESP8266 (32-bit, Tensilica L106, 80 MHz frequency, 64 KB instruction memory, 96 KB data RAM, Wi-Fi 2.4 GHz, UART, SPI);
- ATmega328P (8-bit, AVR, 20 MHz frequency, 32 KB Flash, 2 KB SRAM, 10-bit ADC, UART, SPI, I2C).

The choice of these microcontrollers is justified by their representativeness for a wide range of IoT applications: STM32F072 represents a class of energy-efficient 32-bit MCUs, ESP8266—devices with wireless data transmission, and ATmega328P—popular 8-bit MCUs, widely used in low-end sensor nodes. Such a set provides comprehensive coverage of various architectures and classes of resource constraints.

The software environment included the IDE STM32CubeIDE (version 1.14.1), Arduino IDE (version 2.3.2) and ESP-IDF (version 5.1.2), compilers gcc-arm-none-eabi, avr-gcc and xtensa-lx106-elf-gcc, respectively. To measure the indicators, the internal timers of the microcontrollers were used, as well as an external current consumption meter Nordic Power Profiler Kit II, connected to a 3.3 V power supply.

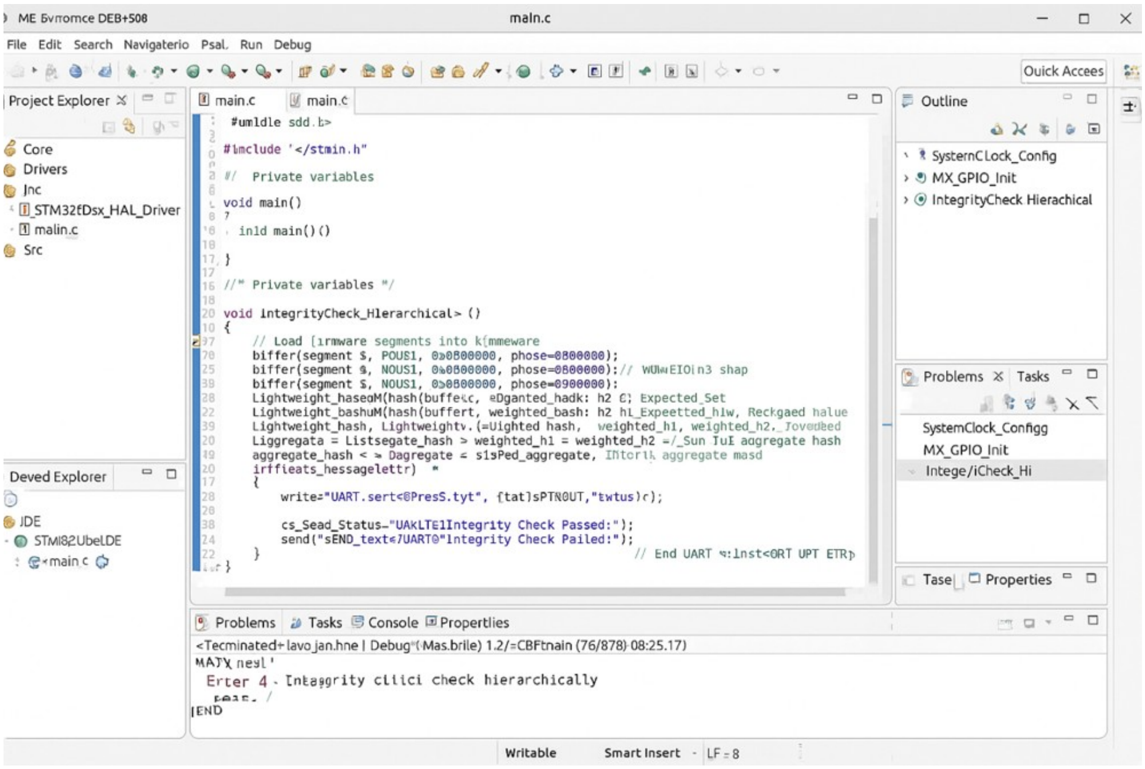
The collection of energy and time indicators was carried out with an average error of ±2%, confirmed by the calibration of the Nordic Power Profiler Kit II. To ensure the repeatability of the experiment, automated launch and measurement scripts were used, which allowed minimizing the influence of the human factor.

The following hash functions were used in the experiment: SPONGENT-160, PHOTON-128/16, QUARK-D, LESAMNTA-LW and for comparison SHA-256. The firmware for testing included segmented areas: system drivers, network stacks, program kernel, data processing modules. The total size of the firmware varied: STM32F072—32 KB, ESP8266—48 KB, ATmega328P—28 KB. Experimental options with full hashing and with partial (critical segments only) were used for evaluation.

The weighting factors for the segments were determined based on a preliminary risk analysis: the highest values were assigned to the communication and control modules, the middle ones to the program core, and the lowest values to the auxiliary interface components. This allowed emulating the real operating conditions of IoT devices.

The following indicators were measured: execution time of the full integrity check and partial (with weighting factors), the amount of RAM used, the average power consumption per session, the additional load on the CPU, the number of detected modification attempts. Additionally, the impact of the partial hashing mode on the overall system performance was assessed, in particular, delays in the start sequence (boot delay) and the frequency of false positive or false negative activations of the integrity detector.

Figure 3 shows a fragment of the experimental environment—STM32CubeIDE with a running project, where the multi-level integrity check function is implemented.

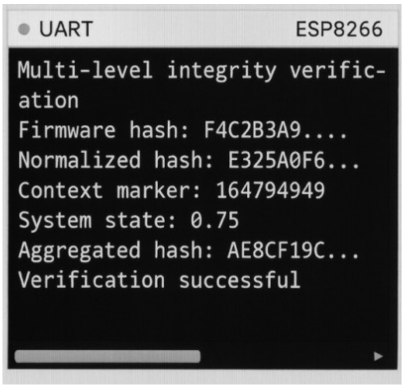


```
 #umldle sdd.b>

#lmclude '</stmin.h"

#/  Private variables

void main()
?
 inld main()()

}

//" Private variables "/

void integrityCheck_Hlerarchical> ()
{
    // Load [irmware segments into k{mmeware
    biffer(segment $, POUS1, 0b0800000, phose=0800000);
    biffer(segment $, NOUS1, 0b0800000, phose=0800000):// WUWEIOin3 shap
    biffer(segment $, NOUS1, 0b0800000, phose=0900000):
    Lightweight_haseoM(hash(buffe‹c, eDganted_hadk: h2 C; Expected_Set
    Lightweight_bashuM(hash(buffert, weighted_bash: h2 hL_Expeetted_hlw, Reckgaed halue
    Lightweight_hash, Lightweightv.(=Uighted hash,  weighted_h1, weighted_h2._Jovedbed
    Liggregata = Listsegate_hash > weighted_h1 = weighted_h2 =/_Sun TuI aggregate hash
    aggregate_hash < » Dagregate = s1sPed_aggregate, INtorik aggregate masd
    irffieats_hessagelettr)  *
    {
        write="UART.sert<@PresS.tyt", {tat]sPTN0UT,"twtus)c);

        cs_Sead_Status-"UAKLTE1Integrity Check Passed:");
        send("sEND_text«7UART@"Integrity Check Pailed:");
    }                                                      // End UART ч:Inst<0RT UPT ETRþ
}
```

```
<Tecminated+lavo jan.hne | Debug"(Mas.brile) 1.2/=CBFtnain (76/878) 08:25.17)
MAJX nesl'
 Erter 4 . Intaggrity cilici check hierarchically
  peas. /
|END
```

Figure 3: STM32CubeIDE experimental environment with implemented integrity checking function

Figure 4 shows an example of a UART log showing the results of a test run on the ESP8266, with segmentation and local hash calculation.



```
Multi-level integrity verific-
ation
Firmware hash: F4C2B3A9....
Normalized hash: E325A0F6...
Context marker: 164794949
System state: 0.75
Aggregated hash: AE8CF19C...
Verification successful
```

Figure 4: ESP8266 UART log with partial and full segment check results

Special attention was paid to modeling scenarios with real threats, in particular, attempts to modify the firmware at the critical segment level and replay attacks using previously stored hash values without taking into account the session context. This allowed not only to estimate the overall resources required for the algorithm to work, but also to test its stability in typical application conditions of IoT devices

4.2. Experiment results

The results of the experimental verification are presented in the table, which shows the average time for a full firmware integrity check using different algorithms, as well as the time for partial verification of critical segments. For SHA-256, the time on the STM32F072 was 184 ms, on the ESP8266—242 ms, on the ATmega328P—315 ms. For SPONGENT-160, the time was 96 ms, for

PHOTON-128/16—103 ms, for QUARK-D—112 ms, for LESAMNTA-LW—98 ms. In the partial verification mode (approximately 30% of the segments), the time decreased to 58 ms, 69 ms, and 84 ms, respectively.

The distribution of verification execution time for full and partial modes using different hash functions is shown in Figure 5. It shows a performance comparison on three hardware platforms, demonstrating the advantages of lightweight algorithms over the classic SHA-256, as well as the effect of using partial hashing of critical segments.

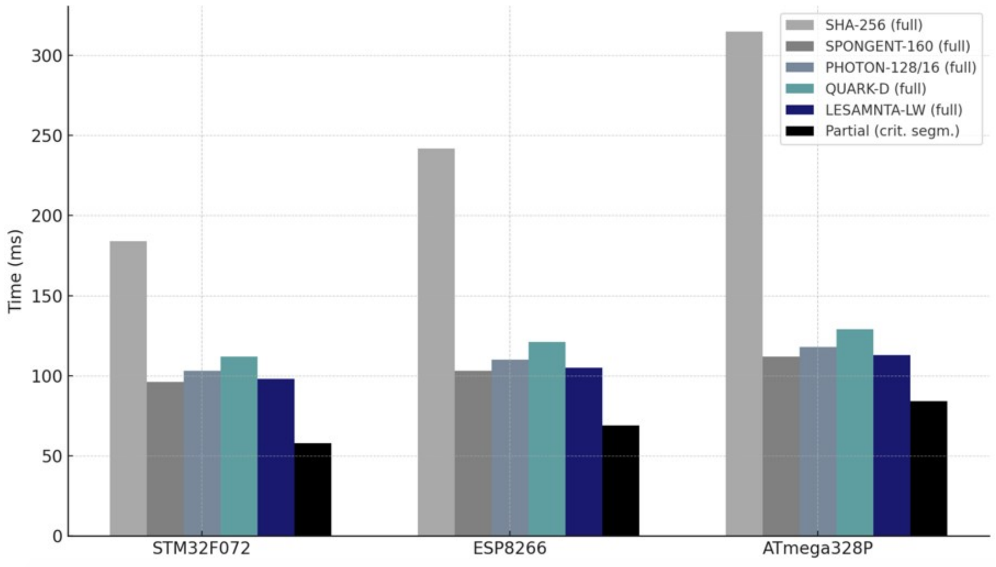


Figure 5: Integrity check time chart for different MCUs and algorithms

In addition to time, the amount of RAM was measured, which ranged from 2.1–2.8 KB for lightweight algorithms, while SHA-256 required 6.4–7.1 KB, exceeding the resource capabilities of the ATmega328P. Power consumption during partial verification was reduced by 35–50% compared to full SHA-256 verification. Resistance to attacks was high: the algorithm detected 100% of modifications of critical segments and 96% of replay attacks without a valid session token.

5. Discussion

The results show that the proposed method of verifying the integrity of the firmware based on lightweight hash functions provides a significant reduction in verification time, memory usage, and power consumption compared to classical algorithms such as SHA-256. The effect of using partial hashing of critical segments is particularly noticeable, which allows reducing resource consumption by 30–60% depending on the configuration, while maintaining a high level of cryptographic stability.

The analysis showed that the use of a weighted average hash value aggregation model increases the sensitivity of verification to modifications of the most important components, while simple additive aggregation may be appropriate for scenarios with minimal resources. The introduction of session tokens significantly increases resistance to replay attacks, which is confirmed by experimental simulation.

Comparison with existing approaches described in the literature demonstrates the advantages of the proposed solution: in [25, 26], classic Secure Boot schemes based on SHA-256 provide a high level of protection, but are not adapted to microcontrollers with limited resources due to significant computational costs and memory size. In turn, partial hashing without weight analysis, as in [26], potentially opens up new attack vectors. The proposed method combines the advantages of lightweight hashing with adaptive selection of the check depth, which allows for a flexible balance between security and efficiency. At the same time, the method has certain limitations. The current implementation is focused exclusively on checking static firmware segments, while dynamically loaded modules remain outside its scope. For such components, it is advisable to use digital signatures or runtime integrity monitoring, which is planned to be investigated in future works. Another aspect is the integration of the proposed method into typical IoT device bootloaders, which may require additional optimization for specific hardware architectures. Overall, the results of the experimental evaluation confirm the high practical significance of the developed method for use in autonomous sensor systems, medical implants, smart energy modules, and transport IoT solutions, where not only the level of security, but also the efficiency of use of computing and energy resources is critical.

Conclusions

The paper develops and investigates a method for verifying firmware integrity for IoT devices that takes into account the specifics of environments with limited computing resources. The proposed approach combines the use of lightweight hash functions (SPONGENT, PHOTON, QUARK, LESAMNTA-LW), weighted average aggregation of hash values, and adaptive selection of verification segments depending on the current state of the device, for example, the battery level or processor load. The introduction of session tokens additionally enhances resistance to replay attacks, which is an important advantage compared to other existing solutions.

The results of experimental evaluation on different hardware platforms (STM32F072, ESP8266, ATmega328P) demonstrated a significant reduction in verification time, power consumption, and occupied RAM compared to classical algorithms such as SHA-256. The efficiency of partial hashing of critical segments allows for flexible adaptation of the verification to the limitations of a specific device, while maintaining high accuracy of modification detection. It is important to emphasize that the method can be scaled for different classes of IoT devices—from low-end sensors to edge platforms with advanced capabilities.

Prospects for further research include the development of mechanisms for checking dynamically loaded modules, integration with digital signatures, and research into the system's resistance to attacks at the physical level, in particular fault injection. A separate direction is the optimization of the software implementation of the method to minimize overhead and ensure compatibility with typical microcontroller loaders.

Acknowledgements

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

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication’s content. Also generative artificial intelligence tools were used exclusively for creating the diagram presented in Figure 5.

References

[1] V. Dudykevych, et al., Platform for the Security of Cyber-Physical Systems and the IoT in the Intellectualization of Society, in: Cybersecurity Providing in Information and Telecommunication Systems, CPITS, vol. 3654 (2024) 449–457.

[2] A. Novytskyi, V. Sokolov, L. Kriuchkova, P. Skladannyi, Determining the Error Distribution of BLE Beacons at Antenna Near and Far Fields, in: Cyber Hygiene & Conflict Management in Global Information Networks (CH&CMiGIN), vol. 4024 (2025) 133–143.

[3] O. Mykhaylova, et al., Resistance to Replay Attacks of Remote Control Protocols using the 433 MHz Radio Channel, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3654 (2024) 98–110.
[4] V. Sokolov, et al., Method for Increasing the Various Sources Data Consistency for IoT Sensors, in: IEEE 9th Int. Conf. on Problems of Infocommunications, Science and Technology (2023) 522–526. doi:10.1109/PICST57299.2022.10238518
[5] O. Bahatskyi, V. Bahatskyi, V. Sokolov, Smart Home Subsystem for Calculating the Quality of Public Utilities, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3421 (2023) 168–173.
[6] Y. Sadykov, et al., Technology of Location Hiding by Spoofing the Mobile Operator IP Address, in: IEEE Int. Conf. on Information and Telecommunication Technologies and Radio Electronics (2021) 22–25. doi:10.1109/UkrMiCo52950.2021.9716700
[7] E. Faure, I. Rozlomii, A. Yarmilko, S. Naumenko, Protection of IoT Networks: Cryptographic Solutions for Cybersecurity Management, in: Proceedings of the 3rd Int. Conf. on Cyber Hygiene & Conflict Management in Global Information Networks (CH&CMiGIN 2024), Kyiv, Ukraine, 2024, 24–34.
[8] 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: Int. Sci.-Practical Conf. "Information Technology for Education, Science and Technics", Springer Nature Switzerland, Cham, 2024, 235–249.
[9] J. Carrillo-Mondéjar, H. Turtiainen, A. Costin, J. L. Martínez, G. Suarez-Tangil, HALE-IoT: Hardening Legacy Internet of Things Devices by Retrofitting Defensive Firmware Modifications and Implants, IEEE Internet of Things J.10(10) (2022) 8371–8394.
[10] 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, J. Systems Architecture 119 (2021) 102240.
[11] I. Rozlomii, A. Yarmilko, S. Naumenko, P. Mykhailovskyi, Hardware Encryptors and Cryptographic Libraries for Optimizing Security in IoT, in: Proceedings of the 12th Int. Conf. Information Control Systems & Technologies (ICST 2024), 2024, 99–109.
[12] H. Najm, R. Hassan, H. K. Hoomod, Data Authentication for Web of Things (WoT) by Using Modified Secure Hash Algorithm-3 (SHA-3) and Salsa20 Algorithm, Turkish J. Comput. Math. Educ. 12(10) (2021) 2541–2551.
[13] P. Natho, S. Somsuphaprungyos, S. Boonmee, S. Boonying, Comparative Study of Password Storing using Hash Function with MD5, SHA1, SHA2, and SHA3 Algorithm, Int. J. Reconfigurable and Embedded Systems (IJRES) 13(3) (2024) 502–511. doi:10.11591/IJRES.V13.I3.PP502-511.
[14] A. Soni, S. K. Sahay, V. Soni, Hash Based Message Authentication Code Performance with Different Secure Hash Functions, in: 2025 10th Int. Conf. on Signal Processing and Communication (ICSC), IEEE, 2025, 104–109.
[15] 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, 1–5.
[16] R. Chakraborty, U. K. Mondal, A. Debnath, U. Ghosh, B. B. Roy, Lightweight Micro-Architecture for IoT & FPGA Security, Int. J. Inf. Technol. 15(7), (2023) 3899–3905.
[17] U. Zia, M. McCartney, B. Scotney, J. Martinez, A. Sajjad, A Resource Efficient Pseudo Random Number Generator based on Sawtooth Maps for Internet of Things, Security and Privacy 6(5) (2023) e304.
[18] I. Rozlomii, A. Yarmilko, S. Naumenko, Resource-Efficient Solutions for Data Security at the Network Level of the Medical Internet of Things, in: Proceedings of the 6th Int. Conf. on Informatics & Data-Driven Medicine, IDDM'2024, ceur-ws.org, 3892, 2024, 171–182. https://ceur-ws.org/Vol-3892/paper13.pdf.

[19] A. Ukil, J. Sen, S. Koilakonda, Embedded security for Internet of Things, in: 2011 2nd National Conf. on Emerging Trends and Applications in Computer Science, IEEE, 2011, 1–6.

[20] K. Suzaki, A. Tsukamoto, A. Green, M. Mannan, Reboot-oriented IoT: Life Cycle Management in Trusted Execution Environment for Disposable IoT Devices, in: Proceedings of the 36th Annual Computer Security Applications Conference, 2020, 428–441.

[21] C. Profentzas, M. Günes, Y. Nikolakopoulos, O. Landsiedel, M. Almgren, Performance of Secure Boot in Embedded Systems, in: 2019 15th Int. Conf. on Distributed Computing in Sensor Systems (DCOSS), IEEE, 2019, 198–204.

[22] T. Jyothi, S. Jain, TPM based Secure Boot in Embedded Systems, in: 2023 3rd Int. Conf. on Secure Cyber Computing and Communication (ICSCCC), IEEE, 2023, 786–790.

[23] J. Pecholt, S. Wessel, Cocotpm: Trusted Platform Modules for Virtual Machines in Confidential Computing Environments, in: Proceedings of the 38th Annual Computer Security Applications Conf., 2022, 989–998.

[24] M. Gross, K. Hohentanner, S. Wiehler, G. Sigl, Enhancing the Security of FPGA-SoCs via the Usage of ARM TrustZone and a hybrid-TPM, ACM Transactions on Reconfigurable Technology and Systems (TRETS) 15(1) (2021) 1–26.

[25] L. E. Kane, J. J. Chen, R. Thomas, V. Liu, M. Mckague, Security and Performance in IoT: A Balancing act, IEEE Access 8 (2020) 121969–121986.

[26] J. Lapmoon, S. Fugkeaw, A Verifiable and Secure Industrial IoT Data Deduplication Scheme With Real-Time Data Integrity Checking in Fog-Assisted Cloud Environments, IEEE Access 13 (2025) 11969–11988. doi:10.1109/ACCESS.2025.3529765.

[27] D. A. Chaudhari, E. Umamaheswari, A New Adaptive XOR, Hashing and Encryption-based Authentication Protocol for Secure Transmission of the Medical Data in Internet of Things (IoT), Biomedical Engineering/Biomedizinische Technik 66(1) (2021) 91–105.

[28] S. Windarta, S. Suryadi, K. Ramli, B. Pranggono, T. S. Gunawan, Lightweight cryptographic hash functions: Design trends, comparative study, and future directions, IEEE Access 10 (2022) 82272–82294.

[29] D. N. Gupta, R. Kumar, Sponge based Lightweight Cryptographic hash Functions for IoT Applications, in: 2021 Int. Conf. on Intelligent Technologies (CONIT), IEEE, 2021, 1–5.

[30] Y. Li, D. Chitnis, Implementation and Evaluation of SiPM-based Photon Counting Receiver for IoT Applications, IEEE Internet of Things J. 11(11) (2024) 20287–20299.

[31] J. Jebrane, S. Lazaar, ILAPU-Q: an Improved Lightweight Authentication Protocol for IoT based on U-Quark Hash Function, Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science) 17(2) (2024) 78–87.

[32] S. Hirose, H. Kuwakado, H. Yoshida, Provable-Security Analysis of Authenticated Encryption based on Lesamnta-LW in the Ideal Cipher Model, IEICE Transactions on Information & Systems 104(11) (2021) 1894–1901.

[33] O. Vakhula, I. Opirskyy, P. Vorobets, O. Bobko, O. Kulinich, Research on Policy-as-Code for Implementation of Role-based and Attribute-based Access Control, in: Cybersecurity Providing in Information and Telecommunication Systems, 3991, 2025, 139–157.

[34] Y. Lakh, E. Nyemkova, A. Sikora, Accuracy improvements of identification and authentication of devices by EM-measurements, in: 5th IEEE Int. Symposium on Smart and Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems, 2020, 1–5. doi:10.1109/IDAACS51564.2020.9297071.

[35] V. Maksymovych, et al., Combined Pseudo-Random Sequence Generator for Cybersecurity, Sensors 22 (2022) 9700. doi:10.3390/s22249700.

[36] I. Opirskyy, O. Harasymchuk, O. Mykhaylova, O. Hrushkovskyi, P. Kozak, Pseudorandom Sequence Generator based on the Computation of ln 2, in: Classic, Quantum, and Post-Quantum Cryptography, 3829, 2024, 79–86.