Securing Mechatronic Control Modules in the Context of Current Cyber Threats

Dănuț-Iulian STANCIU[1], Ioana PETCU[2] , Vlad MANGHER[3]
[1]The National Institute of Research and Development in Mechatronics and Measurement Technique - INCDMTM
[2]National Institute for Research & Development in Informatics - ICI Bucharest
[3]MDM Standard
danut.stanciu@incdmtm.ro, ioana.petcu@ici.ro, vlad.mangher@mdmstandard.ro

Abstract: The increasing integration of connectivity, embedded intelligence, and remote management has turned classical mechatronic systems into complex cyber-physical systems (CPS). While this evolution enables greater efficiency, flexibility and autonomy, it also exposes mechatronic platforms to cyber threats capable of producing direct physical consequences. Unlike traditional IT systems, cyber-attacks on mechatronic systems can compromise control loops, manipulate sensor data, or induce unsafe actuator behavior. This article overviews the cybersecurity threats to modern mechatronic systems, presents the threat landscape specific to embedded control environments, and discusses engineering-oriented security mechanisms suitable for real-time and safety-critical applications. Additionally, we include a practical methodology for securing an integrated mechatronic control module of a concentrating solar system that is connected to the Internet and performs firmware updates over the network. At the same time, our study positions cyber security as a fundamental design constraint in contemporary mechatronics and highlights future research directions relevant to critical infrastructures.
Keywords: cyber security, mechatronics, cyber-physical systems, embedded security, industrial control systems, safety

INTRODUCTION

Mechatronic systems comprise a synergistic integration of mechanical structures, electronic hardware, control algorithms, and embedded software. Originally, such systems operated in isolated environments with limited external connectivity, thus, their exposure to cyber threats was lower than nowadays.

In recent years, driven by the advancements in Industry 4.0 that have led to a digital transformation and an increased demand for remote operation, mechatronic systems have become increasingly networked. Industrial robots, autonomous vehicles, smart actuators, and medical devices now rely on Ethernet-based fieldbuses, wireless communication, cloud services, and over-the-air updates, which

https://doi.org/10.54851/v8i1y202603

are more prone to cyber incidents than before. This transition has had a profound influence on mechatronic platforms such as cyber-physical systems (CPS), since cyber incidents can affect physical processes. Consequently, cybersecurity emerges as a critical concern with implications for safety, reliability, and societal trust—aligning mechatronics firmly within the scope of cybersecurity research.

A cyber-physical system is characterized by the tight coupling between computational elements and physical processes. In mechatronics, this coupling is realized through closed-loop control systems where sensor measurements are continuously processed by embedded controllers to drive actuators.

The key CPS characteristics in modern mechatronics include embedded real-time operating systems, networked control architectures, remote diagnostics and configuration, and integration with supervisory IT systems.

Unlike conventional IT systems, mechatronic CPS operate under strict real-time constraints and are governed by the physical laws. Any cyber compromise propagates beyond data integrity and penetrates into mechanical motion, energy transfer, or force generation.

The cyberthreats affect mechatronic systems since they expose a heterogeneous attack surface that blends IT and operational technology (OT): they target certain components and can be classified accordingly, as:

Embedded Controller Attacks

- Malicious firmware modification
- Secure boot bypass
- Exploitation of debugging interfaces (JTAG, SWD)

Sensor and Actuator Manipulation

- Sensor spoofing or signal injection
- Calibration parameter alteration
- Unauthorized actuator command injection

Industrial Communication Exploitation

- Unauthenticated fieldbus protocols (CAN, Modbus)
- Man-in-the-middle attacks on Ethernet-based automation networks
- Replay and timing attacks affecting control stability

Supply Chain Risks

- Compromised firmware libraries
- Counterfeit or malicious hardware components

These threats are particularly severe because they can induce unsafe physical behaviour without triggering conventional fault detection mechanisms.

To manage the threats and risks specific to industrial and CPS systems, internationally recognized standards and frameworks have been developed:

- IEC 62443 – represents the most comprehensive series of standards for the security of Industrial Automation and Control Systems (IACS) and provides an organized framework of requirements applicable to the entire life cycle of industrial systems: from development, implementation, configuration, and operation to maintenance and updates. IEC 62443 standards define risk models, security levels, network segmentation (zones & conduits) and authentication and authorization requirements adapted to OT environments.
- ISO/IEC 27001 – is an information security management system (ISMS) standard applicable to organizations in any field. Although not specific to CPS or ICS, ISO 27001 provides a robust framework for security governance, risk assessment, and access control that can be integrated with the specific requirements of IEC 62443 for security management in mechatronics.
- NIST – NIST guides (including SP 800-82: Guide to ICS Security and other cybersecurity publications) provide detailed recommendations for protecting industrial cyber infrastructures, including defense strategies, secure configurations, and incident management policies. In practice, organizations use NIST alongside IEC 62443 to build coherent

security programs that are consistent with international best practices.

- In addition to these major frameworks, emerging specialized standards—such as EN 17927 for assessing the security of IoT platforms—can be integrated into the CPS security architecture for component-level trust assessments. Overall, the combined use of these standards ensures holistic coverage of security requirements, both at the technical and organizational levels, and supports the security-by-design approach for critical control systems. Considering the threats mentioned ahead, in the following, we will analyse the practical manner in which an integrated mechatronic system can be secured.

Considering the threats mentioned ahead, in the following, we will analyse the practical manner in which an integrated mechatronic system can be secured.

THE MECHATRONIC APPLICATION DEVELOPED

In the following, the authors will describe a mechatronic application whose securing will later be discussed in the light of the current cyber-threats. The analysed application consists in a command-and-control system for a solar energy capturing device operating based on the principle of concentrating solar rays. This system was developed in the framework of the NUCLEU project – 'PN 23 43 02 01 «New techniques and methods for building solar thermal systems with planar concentrators based on additive processing methods»', which is currently in the Experimental Model stage. The experimental model for the orientation of a Fresnel lens is presented in Figure 1., while the schematic representation of the commands carried out by the electronic control system under analysis is shown schematically in Figure 2.

Figure 1. *Experimental model of the sun tracking system for the planar concentrator*
Source: original, made by authors

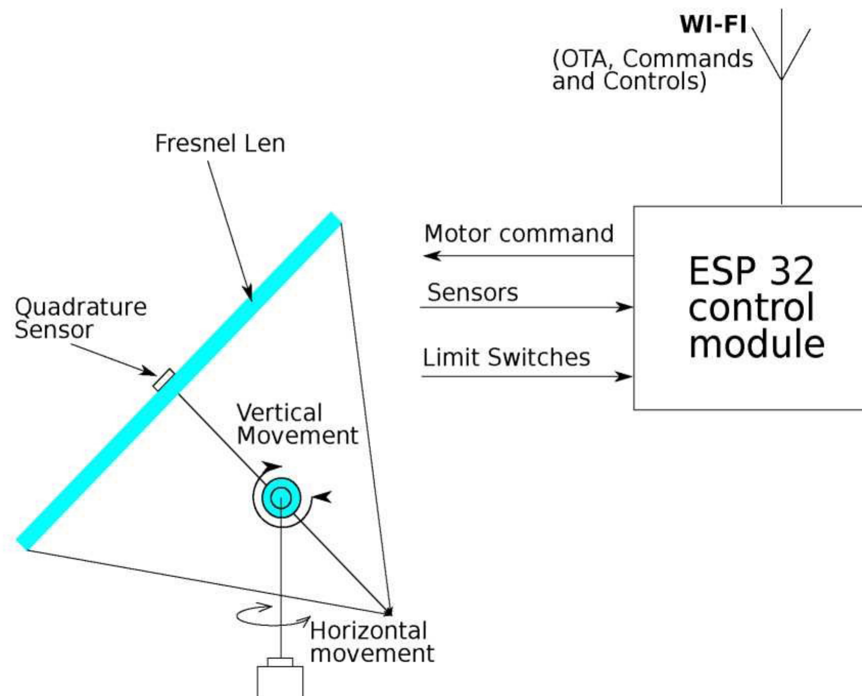


Figure 2. *Schematic representation of the command-and-control elements of the Fresnel planar lens orientation system*
Source: adapted by authors

Mechanical and Control Structure

The mechanical architecture consists of two circular arc ball guides, one horizontal guide used to track the Sun's movement in the East-West direction, and two vertical guides used to track the Sun's vertical movement. On each of the guide rails, the dedicated skate moves and it is driven by a motor-reducer and a rack and pinion mechanism. At the ends of each guide rail, there is a set of stroke limiters consisting of a „soft" magnetic limiter (which sends a signal to the microprocessor) and a „hardware" limiter (which interrupts the electric power supply to the drive motor). The positioning of the lens in the direction of the sun is guided by a four-quadrant sun sensor.

The signals from the four-quadrant sensor cell are read by the microprocessor that controls the displacement motors until the signals from the four integrated sensors become equal.

The integrated control module is built around an ESP32 microprocessor.

The ESP32 processor features a dual-core Xtensa 32-bit LX6 microprocessor operating at an adjustable frequency of up to 240 MHz and integrates Wi-Fi and Bluetooth capabilities.

Table 1. *Technical Specifications of the processor used*

Feature	**Details**
Microprocessor	Dual-core (or single-core in some versions) Tensilica Xtensa 32-bit LX6
CPU frequency	Adjustable, from 80 MHz to 240 MHz
Performance	Up to 600 DMIPS
ULP co-processor	Yes (an ultra-low-power peripheral processor, useful in a deep-sleep mode)
Internal SRAM memory	520 KB
Internal ROM memory	448 KB (for boot and basic functions)
Flash memory	Supports external QSPI Flash memory (typically 4 MB on standard modules, but expandable up to 16 MB)
Wi-Fi connectivity	Standards 802.11 b/g/n (2.4 GHz, up to 150 Mbps)
Bluetooth connectivity	v4.2 BR/EDR and Bluetooth Low Energy (BLE) (dual mode)
Peripheral Interfaces	34 programmable GPIOs, 18 12-bit ADC channels, 2 8-bit DACs, 10 capacitive touch sensors, SPI, I^2C, I^2S, UART, CAN 2.0, Ethernet MAC, Hall sensor, temperature sensor
Energy consumption	Designed for ultra-low power with multiple sleep modes (under 5 µA sleep current)
Security	Secure boot, Flash encryption, hardware cryptographic accelerators (AES, SHA, RSA, RNG)

An important feature of this processor is the possibility of updating the software running on the processor remotely (OTA – Over The Air update), by using the integrated Wi-Fi module. This particularity makes it easier to make changes that often occur in the case of testing experimental models, as is the case presented.

This feature introduces the main security risks generated by: unauthorized modification of the firmware; injection of malicious code through unsecured OTA mechanisms; compromising the control function (incorrect movements, blockages); and man-in-the-middle attacks during software updates.

To protect the system from the threats shown above, a secure update mechanism and a secure boot were chosen (both methods are available in the processor's software libraries, provided by the manufacturer). The secure OTA update provides a firmware distribution channel resistant to interception or tampering attacks, while the secure boot enforces the exclusive execution of cryptographically signed applications, preventing the execution of non-authentic code and maintaining the correct operation of the mechatronic system.

CASE STUDY: SECURING THE FIRMWARE UPDATE VIA HTTPS OTA

The ESP32 Over-The-Air (OTA) mechanism allows firmware to be updated without physical intervention, by downloading a new application image from a remote server and

writing it to a dedicated OTA slot in the flash memory. Using HTTPS introduces a first layer of security by establishing a TLS channel that ensures the confidentiality and integrity of the transfer, thus reducing the risk of interception or modification of the image during transit (man-in-the-middle). In addition, the server certificate verification (either based on a general trust store or a restricted certificate/chain for closed networks) allows the authentication of the distribution endpoint, limiting the possibility of firmware delivery from unauthorized sources. After download, the firmware is validated at the format level (header/compatibility), and activated by changing the boot partition, ensuring a controlled transition to the updated version.

Figure 3 shows schematically how the software update for the ESP32 processor module is performed.

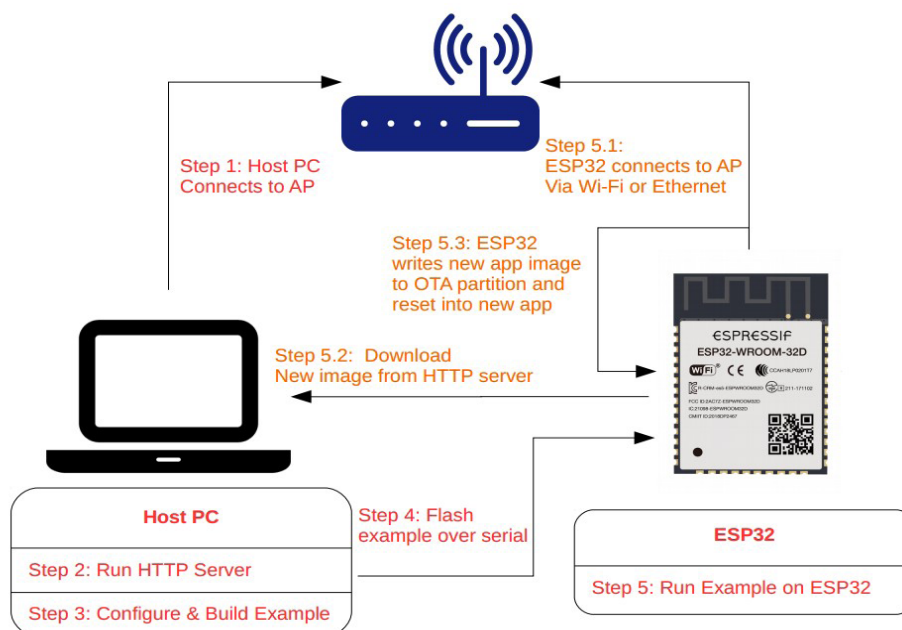


Figure 3. *Schematic representation of a software update for the ESP32 module*
(Source: https://github.com/espressif/esp-idf/blob/v5.5.2/examples/system/ota/ota_workflow.png)

In detail, the stages of updating the firmware are carried out as follows:

Stage 1 – The host computer (on which the guidance system software is built) connects to the Wi-Fi router (via a cable or the Wi-Fi protocol).

Stage 2 – The server on which the newly created image will be stored is configured (it can be on the same computer on which the image is created, or it can be a server in the local network or a server from anywhere in the world connected to the Internet).

Stage 3 – The software is configured and is compiled.

Stage 4 – The ESP module is programmed through the serial connection; the first module programming will contain the over-the-air (OTA) specific components.

Stage 5 – The software is modified and a new software version is created; the new version of the software will be uploaded to the HTTP server.

Stage 5.1 – When the ESP module restarts it connects to the Wi-Fi router and through it to the HTTP server (local or remote).

Stage 5.2 – After connecting to the HTTP server, the OTA module in the processor checks if the image on the server has a

different variant than the current one in the module, and if it is different, it downloads it from the server.

Stage 5.3 – The new software variant is written to the OTA partition, and the module performs a reboot.

At the end of the update, the ESP module will run a new software version.

This update method is at risk of the image being intercepted during the update. To reduce this risk, a secure server with a HTTPS protocol and a TLS-type protocol is used, which ensures the confidentiality and integrity of the transfer, thus reducing the risk of interception or modification of the image during transit ('man-in-the-middle' attack).

Transport Layer Security (TLS) is a standardized cryptographic protocol designed to secure network communications by ensuring the confidentiality, integrity, and authenticity of transmitted data. TLS uses symmetric encryption mechanisms for data flow protection, asymmetric key exchange for secure session establishment, and digital certificates for authenticating the entities involved. In the context of embedded systems, TLS is the foundation of HTTPS communications, preventing interception, modification, or impersonation during data transfer between the device and the server.

Within the TLS protocol, digital certificates play an essential role in the authentication process of the entities participating in the communication. A TLS certificate contains the server's public key and identification information, and it is signed by a trusted Certificate Authority (CA), which allows the client to verify the authenticity of the endpoint it is communicating with. The certificate chain ensures the trust link between the server certificate and a known root authority, preventing impersonation attacks. In embedded systems, certificate selection and management can be achieved either by using a general trust store (for public infrastructures) or by restricted certificates or chains tailored to closed networks, providing increased control over the trust surface.

To secure the connection between the ESP32 and the HTTPS server from which the update is performed and to validate the TLS protocol, 2 types of certificates are used as follows:

Using a General Certificate (Certificate Bundle)

The use of a general certificate (Certificate Bundle) is carried out in the connection used at OTA, by configuring the type variable esp_http_config_t as follows.

To enable the global certificate when using ESP-TLS, simply pass the function pointer to the bundle attach function:

```
esp_tls_cfg_t cfg = {
  .crt_bundle_attach = esp_crt_bundle_attach,
};
```

By setting the variable
.crt_bundle_attach = esp_crt_bundle_attach,

it forces the use of the ESP x509 Certificate Bundle certificate issued by the Mozilla NSS root certificate store and which includes more than 130 certificates that cover more than 99% of the market share.

The list of certificates included in the ESP x509 certificate can be found at wiki.mozilla.org/CA/Included_Certificates

For example (Figure 4), the boot messages (only the sequences important to the current topic) are presented for an application that does not have the boot certificate set and for an application with the variable .crt_bundle_attach = esp_crt_bundle_attach (Figure 5).

```
I (6506) example_common: Connected to example_netif_sta
I (6506) example_common: - IPv4 address: 192.168.0.100,
I (6516) example_common: - IPv6 address: fe80:0000:0000:0000:8a13:bfff:fe68:97c8, type:
ESP_IP6_ADDR_IS_LINK_LOCAL
I (6526) wifi:Set ps type: 0, coexist: 0
I (6526) OTA_ORIENTARE: Starting Orientare OTA
I (6536) OTA_ORIENTARE: OTA started
E (6536) esp_https_ota: No option for server verification is enabled in esp_http_client config.
E (6546) OTA_ORIENTARE: ESP HTTPS OTA Begin failed
I (6556) MAIN: Waiting for OTA result...
```

Figure 4. *Verification of the connection to the update server*
(souce: https://esp32.incdmtm.ro) *without using a TLS connection validation certificate*

As seen in Figure 4, a connection cannot be made without a valid certificate, and the firmware update fails.

In the next test, the variable .crt_bundle_attach = esp_crt_bundle_attach is set. The boot messages in Figure 5 will be displayed.

```
................................................................................................................................
I (7071) OTA_ORIENTARE: Starting Orientare OTA
I (7081) OTA_ORIENTARE: OTA started
I (7091) MAIN: Waiting for OTA result...
I (7271) esp-x509-crt-bundle: Certificate validated
I (7761) esp_https_ota: Starting OTA...
I (7761) OTA_ORIENTARE: Connected to server
I (7761) esp_https_ota: Writing to partition subtype 16 at offset 0x110000
I (7771) OTA_ORIENTARE: Reading Image Description
I (7771) OTA_ORIENTARE: Running firmware version: 1.0.3
I (7781) OTA_ORIENTARE: Verifying chip id of new image: 0
I (25221) esp_image: segment 0: paddr=00110020 vaddr=3f400020 size=270ech (159980) map
I (25281) esp_image: segment 1: paddr=00137114 vaddr=3ffb0000 size=03d0ch ( 15628)
I (25281) esp_image: segment 2: paddr=0013ae28 vaddr=40080000 size=051f0h ( 20976)
................................................................................................................................
```

Figure 5. *Verification of the connection to the update server*
(souce: https://esp32.incdmtm.ro) *by using an esp-x509-crt-bundle certificate for validating the TLS connection*

As can be seen in Figure 5, the certificate used is displayed, the connection to the update server is valid, and the firmware can be updated.

As shown above, the use of the certificate bundle certificate esp-x509-crt-bundle is allowed on nearly 99% of the encrypted servers on the Internet.

Using a Single Certificate

For a connection limited to only one server, for example, the server used in the guidance application (https://esp32.incdmtm.ro), the server's certificate can be directly embedded into the application.

To this end, the certificate fullchain.pem needs to be downloaded from the server (Figure 6) and embedded into the application.

```
-----BEGIN CERTIFICATE-----
MIIFGjCCBAKgAwIBAgISBaQCAvobUekjqLIfA9KYSOJwMA0GCSqGSIb3DQEBCwUA
MDMxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MQwwCgYDVQQD
.................................................................
.................................................................
mYaSipLNgGgiJwunIaJpq+JtEF3MmRhWTtHRLoGn5Qml4vhAe+Va4sbtDy+9OWlH
8c3/8iqatOQnQctnUUCtVQuChDbrou4rALG4rjkg0y4LQnt3wuAsFR5WXwMpMy31
b9XRSZzd5oL+Bj/XQLk=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFBjCCAu6gAwIBAgIRAMISMktwqbSRcdxA9+KFJjwwDQYJKoZIhvcNAQELBQAw
TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh
cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMjQwMzEzMDAwMDAw
WhcNMjcwMzEyMjM1OTU5WjAzMQswCQYDVQQGEwJVUzEWMBQGA1UEChMNTGV0J3Mg
RW5jcnlwdDEMMAoGA1UEAxMDUjEyMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
.................................................................
.................................................................
y5Me5BbrU8973jZNv/ax6+ZK6TM8jWmimL6of6OrX7ZU6E2WqazzsFrLG3o2kySb
zlhSgJ81Cl4tv3SbYiYXnJExKQvzf83DYotox3f0fwv7xln1A2ZLplCb0O+l/AK0
YE0DS2FPxSAHi0iwMfW2nNHJrXcY3LLHD77gRgje4Eveubi2xxa+Nmk/hmhLdIET
iVDFanoCrMVIpQ59XWHkzdFmoHXHBV7oibVjGSO7ULSQ7MJ1Nz51phuDJSgAIU7A
0zrLnOrAj/dfrlEWRhCvAgbuwLZX1A2sjNjXoPOHbsPiy+lO1KF8/XY7
-----END CERTIFICATE-----
```

Figure 6. *The structure of the certificate implemented on the ESP32 microprocessor*

To activate this certificate, into the file CmakeLists.txt is inserted the line:

EMBED_TXTFILES ${project_dir}/server_certs/ca_cert.pem

where

${project_dir}/server_certs/ca_cert.pem

represents the full path to the file that stores the server certificate generated above and shown in Figure 6.

In the file that performs the update of the project (orientare_https_ota.c) we will define the variables:

```
extern const uint8_t server_cert_pem_start[] asm("_binary_ca_cert_pem_start");
extern const uint8_t server_cert_pem_end[] asm("_binary_ca_cert_pem_end");
and the variable that will be used to establish the HTTPS connection:
  esp_http_client_config_t config = {
     .url = CONFIG_EXAMPLE_FIRMWARE_UPGRADE_URL,
     .cert_pem = (char *)server_cert_pem_start,
     .timeout_ms = CONFIG_EXAMPLE_OTA_RECV_TIMEOUT,
     .keep_alive_enable = true,
  };
```

Figure 7. *The file that performs the update of the project*

```
I (6708) esp_https_ota: Starting OTA...
I (6708) OTA_ORIENTARE: Connected to server
I (6708) esp_https_ota: Writing to partition subtype 16 at offset 0x110000
I (6708) OTA_ORIENTARE: Reading Image Description
I (6708) OTA_ORIENTARE: Running firmware version: 1.0.3
I (6718) OTA_ORIENTARE: Verifying chip id of new image: 0
I (31038) esp_image: segment 0: paddr=00110020 vaddr=3f400020 size=270ech (159980) map
I (31088) esp_image: segment 1: paddr=00137114 vaddr=3ffb0000 size=03d0ch ( 15628)
I (31098) esp_image: segment 2: paddr=0013ae28 vaddr=40080000 size=051f0h ( 20976)
I (31108) esp_image: segment 3: paddr=00140020 vaddr=400d0020 size=9f3b8h (652216) map
I (31328) esp_image: segment 4: paddr=001df3e0 vaddr=400851f0 size=11958h ( 72024)
I (31358) esp_image: segment 5: paddr=001f0d40 vaddr=00000000 size=0f240h ( 62016)
I (31378) esp_image: Verifying image signature...
I (31728) esp_image: segment 0: paddr=00110020 vaddr=3f400020 size=270ech (159980) map
I (31778) esp_image: segment 1: paddr=00137114 vaddr=3ffb0000 size=03d0ch ( 15628)
I (31788) esp_image: segment 2: paddr=0013ae28 vaddr=40080000 size=051f0h ( 20976)
I (31798) esp_image: segment 3: paddr=00140020 vaddr=400d0020 size=9f3b8h (652216) map
I (32018) esp_image: segment 4: paddr=001df3e0 vaddr=400851f0 size=11958h ( 72024)
I (32048) esp_image: segment 5: paddr=001f0d40 vaddr=00000000 size=0f240h ( 62016)
I (32068) esp_image: Verifying image signature...
I (32408) OTA_ORIENTARE: ESP_HTTPS_OTA upgrade successful. Rebooting ...
```

Figure 8. *The messages displayed when using the certificate on the server.*

As can be seen in figure 8, the connection to the update server has been established (https://esp32.incdmtm.ro), which shows that the certificate on the microprocessor is valid and the firmware update has been performed.

This method provides a significantly higher level of security than the alternatives. However, it requires periodic updates to both the client-side certificate and the certificate installed on the server.

In practice, server certificates are typically renewed every 3 to 12 months.

For this reason, this approach is generally used in isolated networks that are not connected to the Internet, where certificate updates cannot be performed automatically, as is commonly done for publicly accessible Internet services.

ENSURING APPLICATION INTEGRITY BY SECURE BOOT

The ESP32 platform provides a secure boot mechanism as a method to ensure the integrity and authenticity of the software executed upon booting. Secure boot is based on a cryptographic verification of the application image (and, optionally, of the bootloader) using keys stored securely in the device's non-volatile memory (eFuse). Depending on the desired security level, the ESP32 allows the activation of the secure boot by allowing exclusively running of digitally signed applications ('Require signed app images' option), as well as by extending the check on the bootloader. These mechanisms prevent the execution of unauthorized or modified firmware, thus protecting the system against firmware injection attacks and ensuring a chain of trust since the device boots up.

In the analysed case, of an experimental model, the encryption of the bootloader will greatly complicate the stages of verification, testing, and modification, and therefore, to secure the image of the application, only the digital signature method of the application will be used.

To digitally sign a project with ESP32, the following command is executed in the root of the project

espsecure.py generate_signing_key --version 1 secure_boot_signing_key.pem

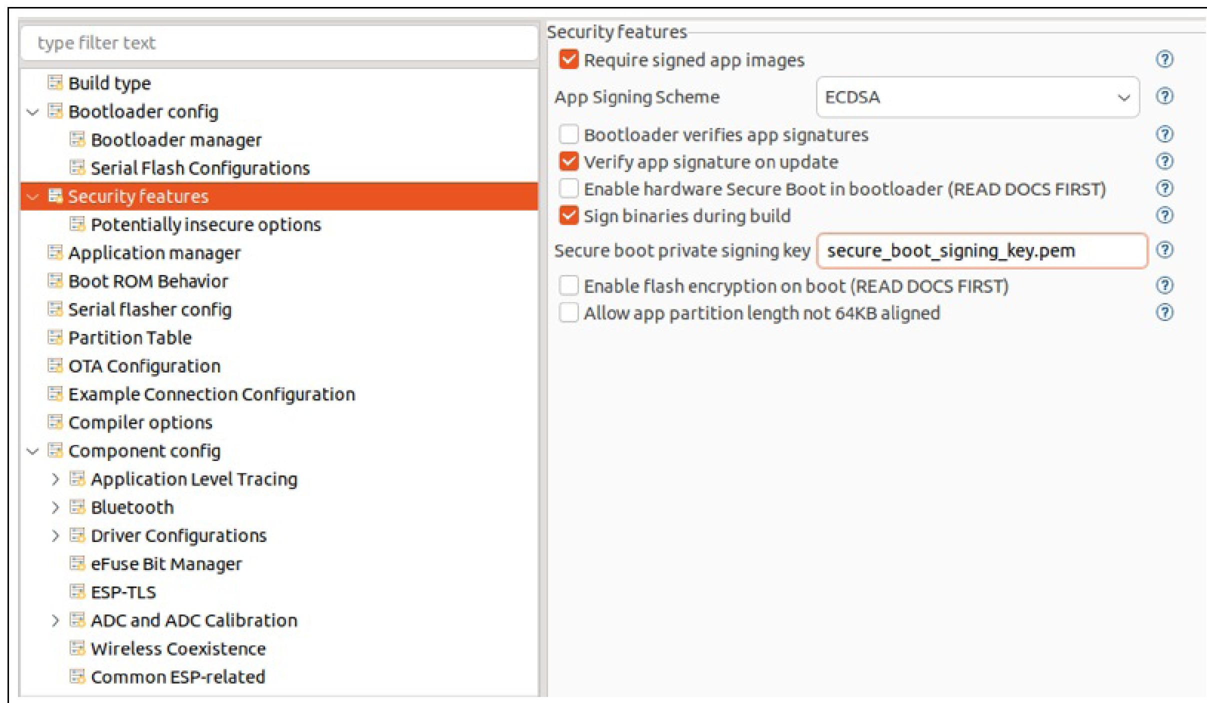


Figure 9. *Selecting the right options for digitally signing the application image*

Figure 9 shows the options that need to be checked for securing the device developed by the authors. They are discussed below.

Require signed app images – the written image must be encrypted.

App Signing Scheme – type of digital signature; in case of the processor chosen and used by the authors, the only possible method is ECDSA (‘**Elliptic Curve Digital Signature Algorithm**’) – which is a **digital signature algorithm** based on elliptic curve cryptography, used for **authenticating and ensuring the integrity** of a firmware image.

Verify app signature on update – when updating through the OTA mechanism, the image is verified by checking the signature.

Sign binaries during build – the created binary (executable) file will be signed by using the key from the „Secure boot private signing key” file, created by the espsecure.py command above.

The encryption and validation process takes place as described below.

In the secure boot mechanism:

- the checksum is calculated for the firmware (e.g. SHA-256);
- the checksum is signed with the ECDSA private key;
- the signature is attached to the image;

at boot, ESP32:

- recalculates the hash,
- verifies the signature with the public key,
- rejects the image if the check fails.

This process prevents running modified firmware, injecting malicious code, and downgrade or replacement attacks.

Figures 10 and 11 show the messages displayed when an unsigned or incorrectly signed image is used (Figure 10) and the messages displayed when an image is correctly signed.

```
I (7244) OTA_ORIENTARE: Reading Image Description
I (7254) OTA_ORIENTARE: Running firmware version: 2.0.3
I (7254) OTA_ORIENTARE: Verifying chip id of new image: 0
I (22724) esp_image: segment 0: paddr=00210020 vaddr=3f400020 size=26d84h
I (22784) esp_image: segment 1: paddr=00236dac vaddr=3ffb0000 size=03d0ch
I (22794) esp_image: segment 2: paddr=0023aac0 vaddr=40080000 size=05558h
I (22804) esp_image: segment 3: paddr=00240020 vaddr=400d0020 size=9f02ch
I (23024) esp_image: segment 4: paddr=002df054 vaddr=40085558 size=115f0h
I (23044) esp_image: Verifying image signature...
E (23044) secure boot v1: image has invalid signature version field 0xffffffff (image
without a signature?)
E (23054) esp_image: Secure boot signature verification failed
I (23054) esp_image: Calculating simple hash to check for corruption...
W (23324) esp_image: image valid, signature bad
E (23324) OTA ORIENTARE: Image validation failed, image is corrupted
E (23324) OTA ORIENTARE: ESP HTTPS OTA upgrade failed 0x1503
I (23324) OTA_ORIENTARE: OTA finish
I (23334) MAIN: OTA finished successfully
I (23334) main_task: Returned from app_main()
```

Figure 10. *Boot messages when using an unsigned application*

It can be seen in Figure 10 that – in the case of using an unsigned image when verifying the signature, the image is rejected and the firmware update is not performed.

```
I (5244) OTA_ORIENTARE: Running firmware version: 2.0.3
I (5254) OTA_ORIENTARE: Verifying chip id of new image: 0
I (25134) esp_image: segment 0: paddr=00210020 vaddr=3f400020 size=270ech
I (25184) esp_image: segment 1: paddr=00237114 vaddr=3ffb0000 size=03d0ch
I (25194) esp_image: segment 2: paddr=0023ae28 vaddr=40080000 size=051f0h
I (25204) esp_image: segment 3: paddr=00240020 vaddr=400d0020 size=9f3b8h
I (25434) esp_image: segment 4: paddr=002df3e0 vaddr=400851f0 size=11958h
I (25454) esp_image: segment 5: paddr=002f0d40 vaddr=00000000 size=0f240h
I (25484) esp_image: Verifying image signature...
I (25834) esp_image: segment 0: paddr=00210020 vaddr=3f400020 size=270ech
I (25894) esp_image: segment 1: paddr=00237114 vaddr=3ffb0000 size=03d0ch
I (25904) esp_image: segment 2: paddr=0023ae28 vaddr=40080000 size=051f0h
I (25904) esp_image: segment 3: paddr=00240020 vaddr=400d0020 size=9f3b8h
I (26124) esp_image: segment 4: paddr=002df3e0 vaddr=400851f0 size=11958h
I (26154) esp_image: segment 5: paddr=002f0d40 vaddr=00000000 size=0f240h
I (26174) esp_image: Verifying image signature...
I (26524) OTA_ORIENTARE: ESP_HTTPS_OTA upgrade successful. Rebooting ...
I (26524) OTA_ORIENTARE: Boot partition updated. Next Partition: 17
I (26524) OTA_ORIENTARE: OTA finish
```

Figure 11. *Boot and update messages when using a signed image*

As can be seen in Figure 11, no error message is displayed, the image signature is updated, and the process runs smoothly, without any errors.

Results: Comparative Analysis of the Implemented Firmware Security Methods

Table 2 shows the advantages and disadvantages of the previously presented security methods.

Table 2. *Comparative matrix of the presented security methods.*

Variant	Security level	Implementation complexity	Impact on maintenance	Practical applicability
Unsecured OTA	Low	Low	Low–Medium	Only for strictly isolated prototypes/ environments
OTA HTTPS + general certificate (bundle/ public CA)	Hugh (for transport & server authentication)	Medium	Low	Very good for modules connected to the Internet
OTA HTTPS + specific certificate (CA/ restricted chain, closed networks)	High–Very High (low trust surface)	Medium–High	Medium	Excellent for industrial/ closed networks
OTA + Secure Boot (signed application)	Very High (runtime integrity/	High	Medium	Recommended for critical systems & field-deployed systems

CONCLUSION

From the analysis of the main security methods presented above, the authors recommend the use of the following security methods:

- For laboratory prototyping: unsecured OTA may be used temporarily, but only in strictly controlled environments.
- For devices connected to a public network: OTA over HTTPS with a general certificate (bundle) should be considered the baseline requirement.
- For devices operating on closed or industrial networks: OTA updates should be delivered over HTTPS using a device-specific certificate with a restricted CA/ chain.
- For critical mechatronic systems (operating high-risk execution systems connected to the internet): it is recommended to use OTA over HTTPS (any variant) combined with Secure Boot (signed application) to ensure an in-depth defence for both transport and execution.

ACKNOWLEDGEMENTS

This study was conducted as part of the project „New techniques and methods for the manufacture of solar thermal systems with flat-plate concentrators using additive manufacturing methods", conducted in the framework of NUCLEU programme, and financed by the Romanian Ministry of Education and Research.

„At the same time, this work was supported by the CERMISO Research Center of INCDMTM – Project Contract no. 159/2017, POC-A.1-A.1.1.1-F Programme."

REFERENCE LIST

Bryan Pearson, Lan Luo, Cliff Zou, Jacob Crain, Yier Jin, et al. Building a Low-Cost and State-of-the-Art IoT Security Hands-On Laboratory. *2nd IFIP International Internet of Things Conference (IFIPIoT), Oct 2019, Tampa, FL, United States,* 289-306, doi:10.1007/978-3-030-43605-6_17. hal-03371609

ESP32 API Reference - Over The Air Updates (OTA) https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/ota.html

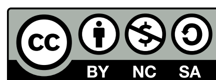
ESP32 Reference Manual – Secure Boot v2 - https://docs.espressif.com/projects/esp-idf/en/stable/esp32/security/secure-boot-v2.html

Petcu, I., Barbu, D.C. (2022) The New Challenges of Romania's Cyber Security Policy. *Romanian Cyber Security Journal,* 4(1), 57-67, doi: 10.54851/v4i1y202207

Petcu, I, Candet, I.B., Stefanescu, c., Gruia, C.I., Craioveanu, V. (2021) Security Risks of Cloud Computing Services from the New Cybernetics' Threats Perspective. *Romanian Cyber Security Journal*, 3(1), 89-97.

Radu, A. F., Petcu, I., Barbu, D.C. (2022) Privacy and security – related challenges of the future EU Digital Identity. *Romanian Cyber Security Journal*, 4(2), 39-52. doi: 10.54851/v4i2y202205

Radu, A. F., Petcu, I. (2021) Intrinsic aspects of e-Government consolidation across the European Union. Case study: Romania. *Romanian Journal of Information Technology and Automatic Control*, 31(4), 83-96. doi: 10.33436/v31i4y202107.



This is an open access article distributed under the terms and conditions of the
Creative Commons Attribution-NonCommercial 4.0 International License.