Ejecución Segura y Verificación de Firmware Cifrado en Tiempo de Ejecución en Entornos IoT

Alejandro Salinas Victorino
Instituto Nacional de Astrofísica, Óptica y Electrónica
Tonantzintla, Puebla, México
alejandro.salinasv@inaoe.mx

Raziel César Campos Sánchez
Instituto Nacional de Astrofísica, Óptica y Electrónica
Tonantzintla, Puebla, México
rcampos@inaoe.mx

José de Jesús Zapata Lara
Instituto Nacional de Astrofísica, Óptica y Electrónica
Tonantzintla, Puebla, México
josedj.zapata@inaoe.mx

***Resumen*—En entornos del Internet de las Cosas (IoT), los dispositivos embebidos basados en microcontrolador como el ESP32 suelen carecer de mecanismos robustos de protección del firmware, exponiéndolos a ataques de clonación, ingeniería inversa y robo de identidad. Este trabajo presenta una arquitectura de ejecución segura de firmware en tiempo de ejecución (run time) basada en MicroPython, donde los módulos funcionales del firmware se almacenan cifrados en memoria flash y son validados mediante Hash Message Authentication Code (HMAC) antes de su ejecución dinámica. La solución garantiza que solo firmware autorizado puede ejecutarse, mitigando así riesgos de modificación o sustitución maliciosa del código. Se desarrolla un ecosistema modular que integra sensores, actuadores y comunicación HTTP, y se valida la propuesta mediante análisis de memoria flash en escenarios comparativos. Los resultados muestran que, a diferencia de una implementación convencional de cargar el firmware en texto claro al dispositivo IoT, la arquitectura propuesta previene la exposición de datos sensibles y refuerza la integridad y autenticidad del firmware, incluso ante ataques con acceso físico al dispositivo.**

***Palabras clave*—Seguridad IoT, ESP32, MicroPython, ejecución en run time, HMAC, firmware cifrado, autenticación de firmware, memoria flash, ingeniería inversa, robo de identidad.**

I. Introducción

En el ámbito del Internet de las Cosas (IoT), proteger el firmware de los dispositivos conectados es una prioridad clave para salvaguardar la seguridad del ecosistema. Dispositivos embebidos como el ESP32 son ampliamente utilizados por su bajo costo, eficiencia energética y conectividad inalámbrica, pero carecen de mecanismos robustos de protección, lo que los hace vulnerables a clonación, ingeniería inversa e inyección de código malicioso. Si bien existen mecanismos como *Secure Boot* y firmas digitales para mitigar estos riesgos, su implementación suele requerir hardware especializado o configuraciones complejas, lo que los hace poco viables en dispositivos de bajo costo o en fases tempranas de desarrollo.

Ante esto, se propone una arquitectura segura de ejecución de firmware en tiempo de ejecución sobre ESP32 usando MicroPython. Esta solución almacena el firmware cifrado en la memoria flash y utiliza códigos HMAC para verificar su integridad y autenticidad antes de su ejecución mediante `exec()`. Solo si la verificación es exitosa, el código se descifra y ejecuta dinámicamente, garantizando que solo firmware autorizado sea cargado.

La propuesta se valida en un sistema modular con sensores y actuadores, comparando un escenario de firmware en texto claro con otro que implementa la arquitectura propuesta. Los resultados muestran que se mitigan ataques de extracción de memoria flash y se protege tanto la lógica del firmware como los datos sensibles, reduciendo riesgos como el robo de identidad. Además, el trabajo se alinea con la guía *NIST-SP-800-193* del NIST, al implementar cifrado de datos críticos y verificación de integridad para mantener los dispositivos en un estado seguro frente a ataques.

El documento se organiza así: la Sección II revisa trabajos relacionados; la Sección III expone los conceptos clave; la Sección IV detalla la metodología de implementación; la Sección V presenta los resultados experimentales; la Sección VI ofrece la conclusión; y la Sección VII plantea futuras mejoras.

II. Trabajo relacionado

La ejecución segura de firmware en dispositivos embebidos ha sido ampliamente abordada desde distintas perspectivas, principalmente mediante mecanismos de *secure boot*, firmas digitales y cifrado. Sin embargo, muchos de estos enfoques dependen de hardware especializado o suponen un entorno de producción controlado, lo cual no siempre es viable en dispositivos de bajo costo o en escenarios dinámicos como el IoT.

Molcut et al. [3], destacan la necesidad de garantizar que los sistemas embebidos ejecuten únicamente firmware autorizado. El trabajo revisa distintas estrategias de protección como el uso de funciones hash, cifrado simétrico/asimétrico y dispositivos criptográficos dedicados. Asimismo, enfatiza que la combinación de mecanismos como verificación de integridad, autenticación y cifrado es esencial para evitar ataques como la sobrescritura de memoria o el uso de firmware malicioso.

Por otro lado, *Wang and Yan* [4] clasifican los principales esquemas de arranque seguro, incluyendo enfoques basados

en hardware (como TPM o TrustZone) y software (como verificación de firmas digitales). Aunque eficaces, estos métodos suelen implicar costos adicionales o altos requerimientos computacionales, lo que limita su adopción en dispositivos con recursos restringidos. En contraste, la solución propuesta en este trabajo busca ofrecer un mecanismo liviano, ejecutado en tiempo de ejecución mediante *MicroPython*, que permite validar y descifrar firmware de forma dinámica usando HMAC.

Younis et al. [5] presentan un enfoque basado en hardware para la validación de la integridad del firmware que garantice el arranque seguro para sistemas IoT con recursos computacionales limitados y para los cuales la variable del costo es importante. La solución propuesta incorpora el uso de PUF's para la identificación de forma única de cada dispositivo IoT y de esta forma establecer una raíz de confianza. Los resultados obtenidos demostraron que la solución propuesta es resistente ante ataques "man-in-the-middle", suplantación de identidad y reproducción de mensajes y en términos de desempeño presenta una carga mínima de recursos de hardware, consumo de energía y latencia.

Feng et al. [1] presentan un esquema de arranque para dispositivos IoT que utiliza Criptografía Basada en Identidad (IBC), el cual ha sido implementado mediante algoritmos criptográficos de curva elíptica. El trabajo propuesto utiliza autenticación bidireccional sin certificado mediante una entidad central que genera y distribuye las claves privadas para los dispositivos y para detectar la manipulación de dispositivos se utiliza la tecnología de PUF's. El esquema propuesto esta integrado por las fases de inicialización del sistema, generación de la imagen de arranque y la validación e inicio del dispositivo.

Finalmente, *Kumar et al.* [2] proponen un marco integral que incluye múltiples capas de autenticación, autorización y cifrado para proteger el proceso de actualización de firmware. Aunque su enfoque está más orientado a escenarios industriales y de infraestructura crítica, comparte la motivación central de asegurar que solo firmware legítimo y validado sea instalado y ejecutado. En ese sentido, la solución aquí presentada se posiciona como una alternativa modular y adaptable para entornos de bajo consumo, reforzando la seguridad desde la capa de ejecución.

III. Preliminares

Esta sección presenta los conceptos esenciales que sustentan la propuesta de ejecución segura de firmware autorizado en dispositivos IoT basados en ESP32 y ejecutados con MicroPython.

- Ejecución en tiempo de ejecución (Runtime Execution): Consiste en interpretar y ejecutar instrucciones dinámicamente durante la operación del sistema, lo cual permite actualizaciones flexibles, carga bajo demanda y ejecución modular del firmware, características especialmente útiles en entornos IoT.
- HMAC para autorización: El Hash Message Authentication Code (HMAC), basado en funciones hash como SHA-256 y una clave secreta, garantiza la integridad y autenticidad del firmware. Solo los módulos con HMAC válido son descifrados y ejecutados.
- Almacenamiento seguro: Hace referencia a la protección de datos sensibles, como claves y configuraciones, en zonas cifradas de la memoria flash del ESP32. Esto evita su exposición ante accesos físicos no autorizados.
- Uso de `exec()` **en MicroPython:** Esta función permite ejecutar código Python desde texto en tiempo de ejecución. En esta propuesta, se emplea para descifrar y ejecutar dinámicamente los módulos del firmware, tras verificar su HMAC, manteniendo así una arquitectura segura y flexible.
- Riesgos de clonación y exposición de información crítica: en entornos IoT, donde los dispositivos suelen estar físicamente expuestos, es común enfrentar ataques como la clonación de firmware y la extracción de datos desde la memoria flash. Un adversario puede replicar el firmware embebido mediante lectura directa o ingeniería inversa, comprometiendo tanto la propiedad intelectual como la integridad del sistema. Asimismo, almacenar claves o configuraciones sensibles sin protección adecuada expone al dispositivo a técnicas como *memory dumping* o *glitching*, vulnerando los principios de confidencialidad, integridad y autenticación.

IV. Metodología

A continuación, se describe la estrategia de diseño e implementación de la arquitectura propuesta para la ejecución segura de firmware autorizado en dispositivos IoT basados en ESP32 utilizando MicroPython. La metodología considera la estructura modular del firmware, el uso de cifrado simétrico y autenticación mediante HMAC, así como la ejecución dinámica del código en tiempo de ejecución.

IV-A. Descripción del firmware

El firmware desarrollado implementa una arquitectura modular orientada a objetos, donde las funcionalidades están encapsuladas en archivos independientes organizados por clases. Esta estructura permite escalabilidad, reutilización y ejecución dinámica de componentes mediante `exec()`. Los módulos principales incluyen: comunicación (`WifiControl.py`, `http_communication.py`), autenticación (`Authentication3.py`), sensores ambientales (`temperature_sensor.py`, `microphone_sensor.py`), procesamiento de datos (`Processing_data.py`), actuadores (`led_semaphore.py`, `IR_send.py`), integración funcional (`Device.py`) y orquestación general (`main.py`).

Adicionalmente se incluye un archibo de configuración `Config.py`, que centraliza parámetros sensibles como pines, llaves de sesión, credenciales Wi-Fi y claves criptográficas AES para las comunicaciones. Esta organización permite mantener los módulos cifrados hasta su ejecución, fortaleciendo la seguridad del sistema.

IV-B. Ejecución segura del firmware en tiempo de ejecución

Se diseñó una arquitectura que permite almacenar y ejecutar el firmware de forma segura. Todos los módulos (excepto `main.py` y `Run_Time_Encryption_Hmac.py`) se encuentran cifrados y verificados mediante HMAC, figura 1. Durante la ejecución:

1. El archivo cifrado es cargado desde memoria flash.
2. Se descifra en RAM usando AES-CBC.
3. Se verifica su integridad con HMAC-SHA256.
4. Si la verificación es exitosa, se ejecuta dinámicamente con `exec()`.

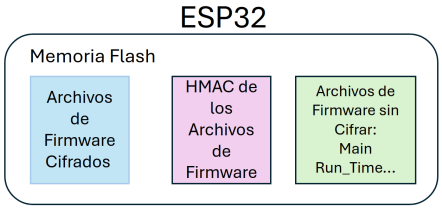

Figura 1. Archivos almacenados en la memoria flash del ESP32.

Este proceso asegura que el firmware nunca reside en texto claro en memoria no volátil, mitigando ataques como clonación o lectura directa de claves criptográficas.

Cabe mencionar que el uso de la función exec() en MicroPython, si bien ofrece flexibilidad para ejecutar código dinámico en tiempo de ejecución, también representa un riesgo significativo si se emplea sin mecanismos de control, ya que permite la ejecución arbitraria de cualquier cadena de texto interpretada como código. Esto podría ser explotado por un atacante para insertar y ejecutar código malicioso, especialmente si el contenido proviene de fuentes no confiables o ha sido modificado en la memoria del dispositivo. Para mitigar este riesgo, la solución propuesta implementa un sistema de validación basado en HMAC (Hash-based Message Authentication Code) que verifica la integridad y autenticidad de cada módulo antes de su ejecución. Solo aquellos fragmentos de firmware cuyo HMAC coincida con el esperado son descifrados y ejecutados mediante exec(), lo que garantiza que únicamente código autorizado y no alterado pueda formar parte del sistema activo, reduciendo significativamente la superficie de ataque y mejorando la seguridad en entornos IoT expuestos.

IV-B1. Cifrado y verificación del firmware:

- Cifrado: Cada archivo en texto claro (por ejemplo, `Device.txt`) es cifrado con AES-CBC y codificado en base64 (`enc_Device.txt`), figura 2.
- HMAC: Se genera un HMAC-SHA256 sobre el archivo original, almacenado como `hmac_Device.txt`, figura 2.
- Verificación: En tiempo de ejecución, se descifran los archivos y el HMAC recibido se compara con el generado localmente. Solo si coinciden, el código es ejecutado, figura 3.

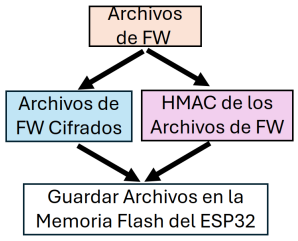

Figura 2. Cifrado de archivos y generación de HMAC del Firmware.

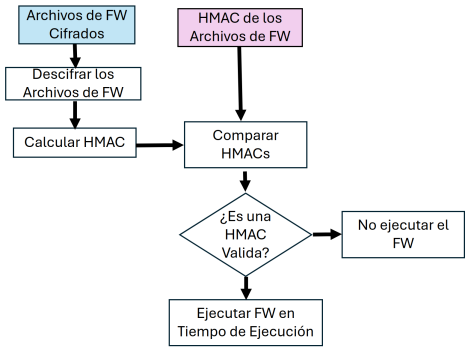

Figura 3. Descifrado y verificación del Firmware para su ejecución.

IV-B2. Componentes en texto claro: Solo dos scripts residen en texto claro en el sistema:

- `Run_Time_Encryption_Hmac.py`: contiene las funciones de cifrado, descifrado y verificación de HMAC.
- `main.py`: orquesta la carga, validación y ejecución de los módulos dinámicos.

En resumen, este enfoque modular garantiza la ejecución exclusiva de firmware autorizado, mitigando riesgos de modificación, ingeniería inversa y clonación, y demostrando su viabilidad en entornos IoT con microcontroladores de recursos limitados como el ESP32.

IV-C. Diferencias Con Secure Boot en ESP32

A diferencia del mecanismo de Secure Boot del ESP32, que requiere configuración en fase de fabricación y permanece estático durante el ciclo de vida del dispositivo, la solución propuesta ofrece una alternativa flexible y adaptable en tiempo de ejecución. Si bien Secure Boot garantiza la integridad del firmware desde el arranque, nuestra arquitectura permite validar y ejecutar módulos dinámicamente, lo que resulta especialmente útil en prototipos o entornos IoT donde el firmware cambia con frecuencia y no se dispone de acceso a eFuses o herramientas de producción avanzadas

V. Resultados y Discusión

En esta sección se presentan los resultados obtenidos a partir de una evaluación comparativa entre dos escenarios implementados sobre el microcontrolador ESP32. El objetivo principal es demostrar la efectividad de la solución propuesta para proteger el firmware y los datos sensibles almacenados

en el dispositivo frente a ataques de extracción directa desde la memoria flash.

- **Escenario 1:** Ejecución del firmware en texto claro, sin mecanismos de cifrado ni verificación de integridad mediante HMAC. En este caso, todos los archivos del sistema (incluyendo configuraciones y credenciales) se almacenan directamente en la memoria flash del ESP32.
- **Escenario 2:** Aplicación de la arquitectura propuesta, en la cual el firmware y los archivos de configuración se almacenan en formato cifrado dentro de la memoria flash, y son verificados y ejecutados dinámicamente en tiempo de ejecución (*run time*) mediante MicroPython.

V-A. Extracción de Memoria Flash y Exposición de Datos Sensibles

Para evidenciar las vulnerabilidades asociadas al almacenamiento en texto claro, se realizó una lectura (dump) completa de la memoria flash en ambos escenarios utilizando la herramienta `esptool.py`, seguida de un análisis con la utilidad `strings` de Linux para la extracción de texto legible. Esta metodología simula un ataque físico orientado a clonar el dispositivo o extraer parámetros críticos de configuración.

```
Password = "password123"
WifiPassword = 'Alex1234'
AES_Password = "Alex1234"
```

Figura 4. Contraseña localizada mediante análisis de memoria flash en el Escenario 1.

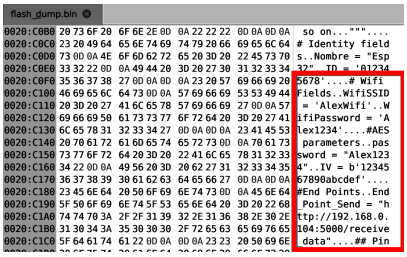

Figura 5. Fragmento de la captura de memoria flash en el Escenario 1, mostrando configuraciones en texto claro.

Como se observa en las figuras 4 y 5, el análisis de la memoria en el escenario 1 revela múltiples elementos sensibles almacenados en texto claro. Entre ellos se encuentran credenciales como contraseñas de red y archivos de configuración completos. Este tipo de exposición representa una amenaza crítica de seguridad, ya que facilita ataques de clonación de firmware, suplantación de identidad de dispositivos e ingeniería inversa del comportamiento del sistema.

Además, el uso de patrones comunes como `Password`, `SSID` o `Token` permite automatizar la extracción de secretos mediante listas predefinidas de búsqueda, lo que aumenta aún más el riesgo de compromiso.

En contraste, la figura 6 presenta los resultados del análisis realizado sobre la memoria flash en el escenario 2. En este caso, la búsqueda del término `Password` no arroja resultados

```
Wifi Invalid Password
Passwords do not match
Password:
```

Figura 6. Resultado del análisis de memoria flash en el Escenario 2: sin exposición de contraseñas.

visibles, lo que confirma que los archivos sensibles han sido correctamente cifrados y que su contenido no es accesible directamente desde la memoria.

Este resultado valida la efectividad de la arquitectura propuesta, la cual mitiga significativamente los riesgos asociados a ataques por lectura directa de la memoria flash, al evitar que secretos criptográficos, credenciales o configuraciones sean expuestos en texto claro.

V-B. Ejecución Segura de Firmware mediante Verificación de HMAC

Además de almacenar el firmware en formato cifrado dentro de la memoria flash del ESP32, el dispositivo IoT implementa un mecanismo de autenticación basado en códigos HMAC (Hash-based Message Authentication Code). Antes de ejecutar cualquier módulo durante el tiempo de ejecución, se realiza una verificación del HMAC correspondiente. Esta validación garantiza que el archivo no ha sido alterado y que proviene de una fuente legítima.

```
This is the run time encryption script.
Main process
the class_name is: ConfigParameters
read file hmac file
HMAC mismatch: File integrity check failed!
```

Figura 7. Resultado de la verificación HMAC de un módulo comprometido.

La figura 7 muestra un ejemplo en el cual la verificación HMAC no es satisfactoria. Este resultado puede deberse a una modificación no autorizada del archivo, corrupción en el proceso de transferencia, o la sustitución maliciosa del módulo. En tales casos, el sistema impide la carga del módulo, evitando su ejecución.

Gracias a la estructura modular del firmware, cada archivo es evaluado de manera independiente. Si alguno de los módulos no pasa la verificación HMAC, el sistema continúa operando con el resto de los componentes válidos, descartando únicamente el archivo comprometido. Esta estrategia permite mantener la operación parcial del dispositivo sin comprometer la seguridad del entorno de ejecución.

De este modo, se asegura que únicamente los módulos que han sido previamente autorizados y cuya integridad ha sido confirmada sean ejecutados. Este proceso actúa como un filtro de seguridad en tiempo de ejecución, garantizando que solo firmware autorizado y verificado forme parte del sistema activo del dispositivo IoT.

V-C. Comparativa de Tiempo de Ejecución y Consumo de Memoria RAM

En esta subsección se presenta una comparativa entre la arquitectura propuesta que incorpora la ejecución y verifi-

cación del firmware en tiempo de ejecución frente a una implementación tradicional, en la que el firmware se ejecuta directamente desde memoria flash sin mecanismos de cifrado ni validación de integridad. Los resultados se obtuvieron mediante pruebas sobre un ecosistema IoT real, diseñado para medir parámetros ambientales como temperatura, humedad y nivel de ruido en un datacenter.

Tabla I
DESEMPEÑO DEL DISPOSITIVO IOT

Arquitectura Propuesta	Tiempo de Sensado	Tiempo de Envío - Recepción de Datos	Uso en Memoria RAM
Sí	3.0 s	10.0 s	98.9 KB
No	3.0 s	10.0 s	46.5 KB

Los parámetros considerados para esta evaluación son:

- Tiempo de sensado: Corresponde al intervalo requerido por el ESP32 para adquirir datos desde sensores ambientales conectados.
- Tiempo de envío y recepción de datos: Tiempo total que tarda en transmitir los datos recolectados hacia un servidor remoto, en este caso desde Zapopan, Jalisco, hasta Ciudad Victoria, Tamaulipas.
- Uso de memoria RAM: Cantidad total de memoria utilizada durante la ejecución completa del firmware bajo cada enfoque.

Como se observa en la tabla I, los tiempos de sensado y de comunicación permanecen constantes en ambos enfoques, lo que demuestra que la solución propuesta no introduce latencia perceptible en la operación del dispositivo. Sin embargo, el consumo de memoria RAM se incrementa significativamente más del doble cuando se ejecuta el firmware de manera cifrada y dinámica. Este aumento es atribuible a las operaciones adicionales necesarias para el descifrado, verificación HMAC y ejecución mediante `exec()` en tiempo de ejecución.

A pesar del mayor uso de memoria, este enfoque proporciona una ventaja considerable en términos de seguridad: impide que el firmware se almacene en texto claro en memoria flash, reduciendo significativamente la superficie de ataque ante intentos de clonación, ingeniería inversa o robo de identidad del dispositivo. Por tanto, el incremento en el uso de recursos es un compromiso razonable a cambio de un entorno de ejecución más robusto y resiliente en aplicaciones IoT expuestas a amenazas físicas.

V-D. Riesgos de ingeniería inversa

Uno de los principales objetivos de la ingeniería inversa es analizar un producto, sistema o componente para comprender su funcionamiento interno, conocer su estructura y, eventualmente, replicarlo o modificarlo. Cuando el firmware de un dispositivo embebido como el ESP32 se encuentra almacenado en texto claro, se expone directamente a este tipo de análisis, lo que representa un riesgo significativo para la confidencialidad del diseño y la propiedad intelectual.

La figura 8 ilustra cómo una porción del firmware almacenado en texto claro puede ser recuperada directamente desde

```
        decipher_text = aes_decrypt.decrypt(cipher_text)
        decipher_text = self.unpad(decipher_text)
        return decipher_text
IoT 2
Authors:
    -- Raziel Campos
    -- Jose Zapata
    -- Alejandro Salinas
from Device import DeviceIoT
# TODO: Move the http class to device, only to test
from http_communication import HttpCommunication
import Config
import json
import hashlib
import time
from Authentication3 import Authentication3
from Processing_data import ProcessingData
def main():
    # The object initialization is once per run
```

Figura 8. Firmware en texto claro recuperado desde la memoria flash.

```
NugFthK3KAoVPV6EFOLyE4RdPY+MPmszgo08CevJyP9ubUBwSUsQqtIQzWMf9fa//Sp
lA6nBmNMgm038xYsDyqVB84b4OLbKkgnnlhwLeRHqz1N3t9bJlBW7JamOYvKDL/oRpe
HihhC9q/uEmEPyR+D5to4uiW1RtGJsniZls8m1UEgeS4ht7SJQIUaM8xEbAuGtayIEJ
0nAB38ZhL1rDi2/tURg5mGe3lzp0AgmZ5L7bcI+q74ucT3GcFmGai9jyXmRc9ZG8YZb
ykcnZNxvgfzHu5AlltXwKQtJq0K5RM0cC54Hxt1U9S9uxewFnPc4gJYzSE1VuKpQofi
NYwjx51fkIS0s+1KxSaVKzc+uxo6xYLjfMwMz4MQDLNLCjY+VKC7a2le3Vfawsbv97o
yt5oeMGMIuUUCqbLuqqDu1lwTRNBiCs43KVzRtMtVgrUPJPLIRHd570Fb3LnIeDzxzB
Wq/6r0/y2WUfPWTt3HUUTRUMA61bLVJ+A8l+QYDFzDrAHdgI+gh1C8PcH4Huq2PVJKM
4JkmVofUWCz9xBkwsFPmaANU0hZMQqt+EHDmUDjAD/ciNhqPviyYBjzoLlWPwty+r6b
YmHyYJMKoZBHNAr9MAT8CGk/AP50crr5RcxaiI/e6S5hte4NLlVa2DYDANehqgd5eDP
/Y3N/2bh5koocbxKfjkX026J37vCR03aRO9u9BzzncDS09coN6C7/lsb0fi4iKsdH6u
NfwVpxRW4GBRzC6P9lBTUH+U/G6bEBpmb0U4tdk3UToaXNZDoi0rXT+PIL29eOLOB9g
uDZj06RKg/TQogVKU8sDQWrcQ1osGVa4a64kB2/CInKpjtpB8A7pM5kgE5+nuayTjw3
yZdd+Zvo83zzH/rx2VL5Cp6r1U7qdOGKoeE5SOlY3kFROM+ozgAqq8UM2hyKkT2rTvp
+o3uga0pD9lPess6UW/w6TLM/y8b0i1VqUn/yXBkHEsRdSURPCbvPIbtvN8Ctpgk802
IC+O8J22mUygtbaK1YPVjENr4vqVDfPLYXGCXWiS+Chueb/eGEN7VPee6OxTPlqlCEq
W97gVMuFqWvGJZNVptRZNNpQglS3AtVd+avA9H8UYBKKH516rmAImPBFYUSTcPTTnGx
bIwL6RXB92aXgtCySRVfh+MNaY8aGbsmIvBQ9Us=
This is the main script to run the IoT project.
Authors:
    -- Raziel Campos
    -- Jose Zapata
    -- Alejandro Salinas
```

Figura 9. Firmware cifrado recuperado en la lectura de la memoria flash.

la memoria flash del dispositivo. En este caso, se identifican fragmentos correspondientes a los módulos de comunicación y sensores, lo cual permite al atacante inferir el comportamiento del sistema y reconstruir su lógica funcional. Este tipo de exposición facilita la ingeniería inversa del firmware y puede derivar en la clonación del dispositivo, modificación del comportamiento o robo de propiedad intelectual.

Por el contrario, la figura 9 muestra el resultado obtenido al aplicar la solución propuesta. En este escenario, únicamente los archivos no cifrados —como el `main.py` y el módulo de descifrado y verificación— son legibles en la memoria flash. El resto del firmware permanece cifrado, lo que impide su interpretación directa incluso si se accede físicamente al contenido de la memoria. Esta medida representa una defensa efectiva contra técnicas de ingeniería inversa basadas en extracción y análisis de firmware.

En resumen, los resultados demuestran que la solución basada en cifrado simétrico y verificación HMAC en tiempo de ejecución incrementa de manera sustancial la protección del dispositivo frente a ataques físicos. Esta capa adicional de seguridad es especialmente relevante en entornos IoT, donde los dispositivos suelen estar desplegados en ubicaciones no controladas y expuestos a actores maliciosos con acceso físico. Es importante mencionar que el proyecto OWASP IoT Top 10 establece que el almacenamiento no seguro y las configuraciones por default representan riesgos de seguridad críticos para los dispositivos IoT y que la base de conocimiento MITRE EMB3D identifica a la falta de protección y verificación en la ejecución del firmware como una de las

principales amenazas a la seguridad de la capa de software de un sistema embebido. Con base en lo anterior, se plantea que el trabajo desarrollado puede ser utilizado como un elemento funcional para el cumplimiento de las directrices que establece la norma ISA/IEC 62443-4-2 sobre los requerimientos técnicos para fortalecer la seguridad de los sistemas industriales para automatización y control, en específico para salvaguardar la confidencialidad de los datos en los dispositivos y la integridad del sistema.

VI. Conclusión

Este trabajo propuso e implementó una arquitectura de ejecución segura de firmware en tiempo de ejecución, orientada a microcontroladores ESP32 que operan bajo MicroPython, sin depender de mecanismos avanzados como *Secure Boot* o módulos criptográficos embebidos. La solución se basa en el cifrado simétrico AES y en la verificación de HMAC, lo que permite almacenar módulos de firmware cifrados y validar su integridad y autenticidad antes de ejecutarlos dinámicamente mediante `exec()` en Micropython.

Este enfoque garantiza que únicamente firmware autorizado puede ser cargado en el sistema, impidiendo la ejecución de código modificado o malicioso. Además, gracias a su estructura modular, se facilita el descarte selectivo de módulos no válidos sin comprometer el funcionamiento global del dispositivo.

Las pruebas demostraron que almacenar el firmware en texto claro expone gravemente a los dispositivos IoT, permitiendo la extracción de contraseñas, llaves criptográficas y configuraciones sensibles desde la memoria flash. Este tipo de vulnerabilidades abre la puerta a ataques de clonación, robo de identidad y suplantación de dispositivos.

En el contexto de la Industria 5.0 —donde la confiabilidad, conectividad y resiliencia son pilares fundamentales— esta arquitectura se posiciona como una solución eficaz para proteger sensores inteligentes, robots colaborativos, sistemas de mantenimiento predictivo y equipos industriales críticos frente a amenazas físicas o lógicas. Su diseño ligero, adaptable y de bajo costo ofrece un camino viable para fortalecer la ciberseguridad en dispositivos embebidos, extendiendo su utilidad más allá del ámbito académico hacia aplicaciones reales en entornos industriales inteligentes.

VII. Trabajo futuro

Este trabajo ha demostrado la viabilidad de ejecutar firmware seguro en dispositivos IoT mediante cifrado simétrico y verificación HMAC en tiempo de ejecución. Sin embargo, existen oportunidades claras de mejora. Una de ellas es la refactorización del archivo `main.py`, que actualmente concentra demasiada lógica, dificultando la adaptabilidad; se propone modularizar aún más su estructura para facilitar la escalabilidad y el mantenimiento.

Asimismo, se plantea incorporar funciones físicas no clonables (PUF) para generar llaves criptográficas únicas sin almacenar datos sensibles en memoria, mitigando riesgos de clonación y robo de identidad. También se propone integrar esta solución en esquemas de actualización segura OTA, particularmente en escenarios distribuidos como los de la Industria 5.0, gracias a la portabilidad de su diseño modular y orientado a objetos.

Finalmente, se propone evaluar la eficiencia del sistema bajo distintas condiciones operativas, como variaciones en el tamaño del firmware, número de módulos cifrados y tiempos de respuesta en redes con conectividad limitada. Estas pruebas permitirán ajustar parámetros de diseño y validar la escalabilidad de la solución en entornos reales.

VIII. Agradecimientos

Agradecemos a SECIHTI por el apoyo financiero brindado mediante becas de posgrado, al INAOE por proporcionar un entorno académico de excelencia, y al cuerpo docente del programa de Maestría en Ciencias y Tecnologías de Seguridad por su valiosa formación. Reconocemos especialmente al profesor de la asignatura de *Internet de las Cosas* por su orientación, así como a los autores y desarrolladores de recursos técnicos utilizados en este proyecto.

Referencias

[1] Feng, Yunlong, Linhai Liu, Dong Liang, Qian Chen, Yongchuan Zhou, Zhihao Wang y Linhai Wu: *IBCEB: A bi-authentication Secure Boot Scheme for IoT device based on IBC*. En *2023 3rd International Conference on Electronic Information Engineering and Computer Science (EIECS)*, páginas 1314–1318, 2023.

[2] Kumar, Sudeendra K., Sauvagya Sahoo, Krishna Kiran, Ayas Kanta Swain y K.K. Mahapatra: *A Novel Holistic Security Framework for In-Field Firmware Updates*. En *2018 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*, páginas 261–264, 2018.

[3] Molcut, Alex Ionut, Septimiu Lica y Ioan Lie: *Cybersecurity for Embedded Systems: A review*. 2022 International Symposium on Electronics and Telecommunications (ISETC), 2022.

[4] Wang, Rui y Yonghang Yan: *A Survey of Secure Boot Schemes for Embedded Devices*. En *2022 24th International Conference on Advanced Communication Technology (ICACT)*, páginas 224–227, 2022.

[5] Younis, Mohamed, Mohammad Ebrahimabadi, Suhee Sanjana Mehjabin, Emily Pozniak, Tamim Sookoor y Naghmeh Karimi: *LiSB: Lightweight Secure Boot and Attestation Scheme for IoT and Edge Devices*. IEEE Transactions on Information Forensics and Security, páginas 1–1, 2025.