Mondragon Unibertsitatea
Escuela Politécnica Superior

TFM

TRABAJO DE FIN DE MÁSTER

Plataforma OTA segura y escalable para dispositivos IoT con criptografía ligera y post-cuántica

Díez-Andino Herrera, Álvaro

2025-2026

Máster en análisis de datos, ciberseguridad y computación en la nube

Plataforma OTA segura y escalable para dispositivos IoT con criptografía ligera y post-cuántica

Trabajo Fin de Máster presentado en: Escuela Politécnica Superior de MU

Para la obtención del título de: Máster en análisis de datos, ciberseguridad y computación en la nube

Autor/a: Díez-Andino Herrera, Álvaro

Director/a: Iñaki Garitano

Tutor/a: Haritz Saiz

Empresa/Organización en la que se ha realizado el proyecto: Ikerlan S. Coop.

DECLARACIÓN DE ORIGINALIDAD

Yo Díez-Andino Herrera, Álvaro

Declaro que este Trabajo de Fin de Máster es original, fruto de mi trabajo personal, y que no ha sido previamente presentado para obtener otro título o calificación profesional. Las ideas, formulaciones, imágenes, ilustraciones tomadas de fuentes ajenas han sido debidamente citadas y referenciadas.

Resumen

El Trabajo de Fin de Máster propone el diseño, implementación y evaluación de una plataforma de actualización remota (OTA) segura y escalable para dispositivos IoT. La motivación principal radica en la elevada cantidad de dispositivos que sufren por vulnerabilidades y la falta de actualizaciones para solucionarlas, lo que crea vectores de ataque a gran escala. Esta plataforma permite la distribución de actualizaciones firmadas y cifradas a flotas de dispositivos para garantizar la autenticidad, integridad y confidencialidad del software desplegado.

El sistema está pensado para todo tipo de escenarios: desde dispositivos sencillos con pocos recursos hasta situaciones críticas que necesitan protegerse de futuros ordenadores cuánticos. Por eso, soporta dos tipos de criptografía: algoritmos post-cuánticos y algoritmos ligeros (lightweight cryptography), ideales para equipos con poca memoria y batería.

El trabajo incluye un análisis del estado de la práctica sobre actualizaciones OTA en IoT y de las necesidades de seguridad actuales, mostrando la importancia de incorporar criptografía ligera para dispositivos con recursos limitados y la opción de soluciones resistentes a la computación cuántica cuando el contexto lo requiera. Además, se desarrolla un prototipo de plataforma, desplegable en la nube y diseñado para escalar horizontalmente.

Palabras clave: OTA, IoT, actualizaciones seguras, post-cuántico, criptografía ligera, firma digital, cifrado.

Impacto en los Objetivos de Desarrollo Sostenible (ODS): ODS 9 (Industria, Innovación e Infraestructura), ODS 12 (Producción y Consumo Responsables).

Laburpena

Master Amaierako Lan honek IoT gailuetarako OTA (over-the-air) eguneratze-plataforma segurua eta eskalagarri baten diseinua, inplementazioa eta ebaluazioa proposatzen du. Motibazio nagusia da ahultasunak dituzten eta eguneratzerik jasotzen ez duten gailu ugariena, eta horrek erasotze-bideak sortzen ditu eskala handian. Plataforma honek eguneratze sinatuak eta zifratutakoak banatzeko aukera ematen du, sistema batean instalatutako softwarearen jatorria, osotasuna eta konfidentzialtasuna bermatzeko.

Baliabide mugatuak dituzten gailuetarako eta etorkizuneko eraso kuantikoen aurrean erresistentzia behar duten egoeretarako egokituz, sistemak bi kriptografia-familiari laguntza eskaintzen die: kuantumaren aurkako erresistentzia duten zifratze eta sinadura algoritmoak, eta memoria, CPU eta energia murrizketak dituzten gailuentzako optimizatutako 'lightweight' kriptografia algoritmoak.

Lanak OTA eguneratzeei buruzko egungo praktiken analisia eta segurtasun-beharren azterketa barne hartzen ditu, baliabide mugatutako gailuentzako kriptografia arina sartzeko garrantzia nabarmentzen du eta beharrezkoa denean kuantumaren aurkako soluzioak aukeratzearen garrantziaz mintzo da. Gainera, hodeian martxan jar daitekeen eta horizontalki eskalatzeko diseinatutako plataforma prototipo bat garatu.

Gako-hitzak: OTA; IoT; eguneratze seguruak; post-kuantikoa; kriptografia arina; sinadura digitala; zifratzea.

Garapen Jasangarriko Helburuetan (GJH) eragina: GJH 9, GJH 12.

Abstract

This Master's thesis proposes the design, implementation, and evaluation of a secure and scalable over-the-air (OTA) update platform for IoT devices. The main motivation is the large number of devices that suffer from vulnerabilities and are not updated, creating large-scale attack vectors. The platform enables distributing signed and encrypted updates to device fleets to ensure the authenticity, integrity, and confidentiality of deployed software.

To accommodate both resource-constrained devices and scenarios requiring resilience against future quantum attacks, the system supports two families of cryptographic algorithms: post-quantum resistant encryption and signing algorithms, and lightweight cryptography algorithms optimized for devices with limited memory, CPU, and energy.

The work includes an analysis of current OTA practices and security needs, highlighting the importance of incorporating lightweight cryptography for constrained devices and offering post-quantum options when the context requires it. Additionally, a cloud-deployable prototype platform is developed, designed for horizontal scalability.

Keywords: OTA; IoT; secure updates; post-quantum; lightweight cryptography; digital signatures; encryption.

Sustainable Development Goals (SDG) impact: SDG 9 (Industry, Innovation and Infrastructure), SDG 12 (Responsible Consumption and Production).

Agradecimientos

Agradezco especialmente a mi tutor/a por su apoyo y guía durante la realización de este Trabajo de Fin de Máster.

También quiero agradecer a la empresa/organización que me ha dado la oportunidad de desarrollar este proyecto en un entorno real.

Finalmente, gracias a mi familia y amigos por su comprensión y apoyo incondicional.

Díez-Andino Herrera, Álvaro

Arrasate-Mondragón, 2025-2026

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

AEAD Authenticated Encryption with Associated Data (Cifrado Autenticado con Datos Asociados)

AES Advanced Encryption Standard (Estándar de Cifrado Avanzado)

API Application Programming Interface (Interfaz de Programación de Aplicaciones)

ASCON Algoritmo de criptografía ligera estandarizado por NIST

CPU Central Processing Unit (Unidad Central de Procesamiento)

CRA Cyber Resilience Act (Ley de Resiliencia Cibernética de la UE)

DDoS Distributed Denial of Service (Denegación de Servicio Distribuida)

DFU Device Firmware Update (Actualización de Firmware del Dispositivo)

ECDSA Elliptic Curve Digital Signature Algorithm (Algoritmo de Firma Digital de Curva Elíptica)

ICS Industrial Control Systems (Sistemas de Control Industrial)

IoT Internet of Things (Internet de las Cosas)

LWC Lightweight Cryptography (Criptografía Ligera)

ML-DSA Module-Lattice-Based Digital Signature Algorithm (Algoritmo de Firma Digital basado en Redes de Módulos, antes Dilithium)

NIST National Institute of Standards and Technology (Instituto Nacional de Estándares y Tecnología de EE.UU.)

ODS Objetivos de Desarrollo Sostenible

OTA Over-The-Air (Actualización remota por aire)

PC Personal Computer (Ordenador Personal)

PQC Post-Quantum Cryptography (Criptografía Post-Cuántica)

RAM Random Access Memory (Memoria de Acceso Aleatorio)

RSA Rivest–Shamir–Adleman (Algoritmo criptográfico de clave pública)

SDK Software Development Kit (Kit de Desarrollo de Software)

SPHINCS+ Esquema de firma digital post-cuántico basado en funciones hash

TFM Trabajo de Fin de Máster

TUTK ThroughTek Kalay (Plataforma IoT)

UE Unión Europea

AEAD Authenticated Encryption with Associated Data (Cifrado Autenticado con Datos Asociados)

1. Introducción

El Internet de las Cosas (IoT) ha transformado radicalmente la forma en que los dispositivos interactúan con el entorno físico y digital. Sin embargo, este crecimiento exponencial ha expuesto una brecha crítica: la falta de mecanismos robustos para mantener la seguridad de estos dispositivos a lo largo de su ciclo de vida operativo. Esta introducción presenta el contexto del problema, examina el estado actual de la investigación y la práctica industrial, y define los objetivos de este Trabajo de Fin de Máster.

1.1 Problemática

El ecosistema IoT ha experimentado un crecimiento sin precedentes, con miles de millones de dispositivos conectados desplegados en entornos industriales, domésticos, sanitarios y de infraestructuras críticas. A pesar de este despliegue masivo, la seguridad no ha evolucionado al mismo ritmo. Una cantidad alarmante de dispositivos permanece sin actualizar debido a la ausencia de mecanismos efectivos de actualización remota, convirtiéndose en vectores de ataque a gran escala.

Las actualizaciones OTA (Over-The-Air) inseguras, o directamente inexistentes, son uno de los mayores riesgos en el IoT. Datos de la industria [2] dicen que cerca del 20 % de las organizaciones han sufrido ataques que venían de dispositivos IoT comprometidos, y que la mayoría de las empresas tienen un riesgo importante por culpa de aparatos sin parchear.

Este problema se complica por dos retos técnicos básicos:

1. **Recursos limitados:** Los dispositivos IoT son muy variados, desde chips muy simples de 8 bits con poquísima memoria hasta sistemas más potentes. Esta variedad hace que no se puedan usar las mismas soluciones de seguridad que usamos en ordenadores normales.
2. **La amenaza cuántica:** Los algoritmos usados actualmente para firmar actualizaciones podrían ser vulnerables a los ordenadores cuánticos del futuro. Como los dispositivos IoT suelen durar muchos años, es buena idea pensar en esto desde el principio para los casos donde la seguridad a largo plazo sea clave.

Si a esto se le suma la cantidad de dispositivos que hay, las actualizaciones poco seguras y las amenazas futuras, tenemos un escenario de riesgo importante. Por eso hace falta buscar soluciones nuevas, aprovechando lo que dice la investigación y la experiencia de la industria.

Además, hay que tener en cuenta que los dispositivos IoT no son ordenadores ni teléfonos. Un sensor industrial, una bombilla inteligente o un monitor de salud tienen restricciones de hardware muy diferentes: procesadores de bajo consumo, memoria limitada y fuentes de energía restringidas. Aplicar los mismos mecanismos de actualización que usamos en un ordenador simplemente no funciona, y eso ha llevado a prácticas inseguras: fabricantes que envían actualizaciones sin cifrar, dispositivos que aceptan firmware sin verificar su origen, o productos que salen al mercado sin capacidad de actualización. Estos problemas han facilitado la creación de botnets y ataques DDoS a gran escala en el pasado, por ejemplo, Mirai [3]. Estudios académicos han identificado además la presencia de SDKs y flujos de actualización inseguros que amplían la superficie de ataque a millones de dispositivos [4].

Por otra parte, los dispositivos IoT tienen ciclos de vida larguísimos, a veces de décadas, lo que dificulta mucho la gestión de vulnerabilidades y actualizaciones con el tiempo [5]. Durante ese periodo, aparecen nuevas vulnerabilidades, se descubren fallos en algoritmos criptográficos y la amenaza de la computación cuántica pone en evidencia la necesidad de pensar en opciones resistentes a largo plazo. Todo esto subraya la urgencia de adoptar un método centralizado y auditable de gestión de actualizaciones que permita controlar despliegues, gestionar rollbacks y garantizar la ciberresiliencia.

Este trabajo se centra explícitamente en la distribución de actualizaciones remotas para dispositivos IoT que ejecutan Linux embebido, el entorno mayoritario en dispositivos industriales y comerciales. «Linux embebido» hace referencia a distribuciones y stacks (Buildroot, Yocto, u-boot, etc.) y a imágenes rootfs y toolchains optimizados para entornos con recursos limitados; no debe confundirse con una instalación de Linux de escritorio o servidor con servicios y recursos plenos. Por tanto, las adaptaciones que este entorno exige son tanto de naturaleza operativa como de límite de recursos.

1.2 Motivación

El ecosistema de los dispositivosIoT enfrenta una discrepancia crítica entre las capacidades teóricas y la implementación real. Si bien la literatura académica demuestra la viabilidad de sistemas de actualización seguros y eficientes, la realidad industrial muestra una adopción limitada de estas tecnologías avanzadas. Los frameworks comerciales predominantes continúan dependiendo de esquemas criptográficos tradicionales, sin integrar algoritmos ligeros para dispositivos restringidos ni prepararse para la amenaza post-cuántica.

Esta brecha tecnológica, sumada a la persistencia de vulnerabilidades en dispositivos desplegados, fundamenta la necesidad de una nueva plataforma. El presente trabajo busca demostrar la viabilidad técnica de integrar herramientas de actualización maduras, como SWUpdate, con criptografía de vanguardia (ligera y post-cuántica) en un entorno de gestión centralizado y escalable.

Adicionalmente, el marco regulatorio actual impulsa esta transformación. La Cyber Resilience Act (CRA) de la Unión Europea [6] y estándares como la IEC 62443 (requisito CR 3.10) [7] establecen la gestión segura del ciclo de vida y las actualizaciones como requisitos obligatorios. Esta propuesta facilita el cumplimiento normativo al proporcionar una infraestructura auditable y robusta para la gestión de actualizaciones.

1.3 Estado del arte

Esta sección analiza sistemáticamente la literatura académica y las soluciones tecnológicas existentes para identificar las contribuciones científicas, las limitaciones de las implementaciones actuales, y las brechas que justifican la propuesta de este trabajo. El análisis se estructura en cuatro partes: evidencia empírica sobre vulnerabilidades, plataformas OTA existentes, familias de algoritmos criptográficos emergentes, y análisis crítico de las limitaciones identificadas.

1.3.1 Panorama de vulnerabilidades en dispositivos IoT

Es evidente que la seguridad en IoT es un problema grave, tanto si miramos informes de la industria como estudios académicos. Todos coinciden en que hacen falta soluciones sólidas para actualizar el firmware.

Evidencia desde la industria:

Los informes de la industria pintan un panorama preocupante. Se estima que más de 5.600 millones de dispositivos IoT serán vulnerables en los próximos años, sobre todo con la llegada del 5G y el crecimiento masivo de dispositivos conectados [8].

Reportes de seguridad [2] evidencian que aproximadamente el 20 % de las organizaciones han detectado ataques basados en dispositivos IoT en sus infraestructuras, y que la gran mayoría de entornos corporativos presentan exposición significativa a riesgos derivados de dispositivos comprometidos o sin actualizar. Esta situación refleja una superficie de ataque en constante expansión, donde la falta de procedimientos de mantenimiento y actualización constituye una debilidad estructural.

Los informes de seguridad destacan consistentemente que los *mecanismos de actualización inseguros* y el *firmware/software obsoleto* se encuentran entre las 10 principales vulnerabilidades de IoT [9]. La falta de aplicación de parches y la antigüedad del código base de muchos dispositivos representa un punto de entrada preferente para los atacantes, siendo a menudo el eslabón más débil de la cadena de seguridad.

Un ejemplo particularmente preocupante de vulnerabilidades en componentes de terceros se reveló en mayo de 2024, cuando se identificaron fallos críticos en la plataforma IoT ThroughTek Kalay (TUTK), afectando a más de cien millones de dispositivos a nivel global, incluyendo cámaras de vigilancia y sistemas de seguridad. La explotación en cadena de estas vulnerabilidades permite comprometer completamente el dispositivo, subrayando cómo un fallo en un componente de terceros puede tener un impacto masivo en todo el ecosistema de productos [10].

Adicionalmente, el firmware desactualizado continúa siendo el principal motor de las botnets IoT, como Mirai, que explotan credenciales por defecto o vulnerabilidades conocidas para reclutar dispositivos y lanzar ataques de denegación de servicio distribuido (DDoS) masivos. Incidentes recientes han documentado ataques DDoS sin precedentes impulsados por routers y dispositivos IoT comprometidos [3], demostrando que el problema no solo persiste, sino que se amplifica con el crecimiento del ecosistema.

Evidencia desde la literatura académica:

Desde una perspectiva más técnica, el estudio «AoT: Attack on Things» de Ibrahim et al. [4] ofrece datos empíricos alarmantes sobre la prevalencia de SDKs vulnerables en el ecosistema de actualizaciones OTA. Los autores analizaron 23 dispositivos IoT comerciales y sus aplicaciones móviles asociadas, identificando seis SDKs de actualización de firmware (DFU, Device Firmware Update) que presentan vulnerabilidades críticas. Mediante un análisis automatizado a gran escala, el estudio reveló que 1,356 aplicaciones disponibles en Google Play Store dependen de estos SDKs vulnerables y que, en conjunto, estas aplicaciones gestionan al menos 61 modelos de dispositivos IoT ampliamente distribuidos. Esta cadena de dependencias inseguras subraya la magnitud del problema: millones de dispositivos en el campo son potencialmente explotables a través de vectores de actualización comprometidos.

Investigaciones recientes sobre IoT industrial [5] confirman estos problemas: firmware que raramente se actualiza tras el despliegue, mecanismos de actualización inseguros, y dispositivos con ciclos de vida de décadas sin mantenimiento adecuado. Todo esto subraya la urgencia de plataformas automatizadas de actualización segura que puedan operar de forma continua a lo largo de toda la vida útil del dispositivo.

La evidencia convergente desde múltiples fuentes establece que las vulnerabilidades en mecanismos de actualización no son casos aislados, sino un problema sistémico que afecta a millones de dispositivos desplegados. Esta constatación motiva el análisis de las soluciones tecnológicas existentes para determinar si abordan adecuadamente estos desafíos.

1.3.2 Plataformas OTA existentes: análisis comparativo

En respuesta a las necesidades documentadas de actualización segura, la comunidad de código abierto y diversos proveedores comerciales han desarrollado plataformas especializadas. Sin embargo, el análisis comparativo revela limitaciones significativas en relación con los requisitos identificados.

SWUpdate [11] es un framework de código abierto para actualizaciones en sistemas Linux embebidos, con soporte para actualizaciones atómicas (esquemas A/B) y firmas RSA/ECDSA, pero sin gestión centralizada de flotas. En resumen, es un software capaz de instalar las actualizaciones de forma segura en el dispositivo, pero no proporciona una solución completa para la gestión de actualizaciones a gran escala. La gran ventaja de SWUpdate es su adaptabilidad gracias a los modulos y su customización.

Eclipse hawkBit [12] ofrece gestión backend de actualizaciones OTA a escala empresarial con rollout progresivo, aunque la seguridad criptográfica depende de la implementación del cliente.

Mender [13] integra cliente y servidor para actualizaciones OTA con gestión de flotas, pero utiliza exclusivamente algoritmos tradicionales (RSA y ECC) sin soporte para criptografía ligera ni post-cuántica.

Balena [14] proporciona gestión de flotas basada en contenedores Docker, introduciendo overhead significativo que lo hace inadecuado para dispositivos muy restringidos.

extbfRAUC [15] es similar a SWUpdate en funcionalidades de actualización atómica, pero tampoco

incluye familias de algoritmos criptográficos avanzadas.

La Tabla 1.1 resume las características principales de estas plataformas.

Tabla 1.1: Comparativa de plataformas OTA existentes

Plataforma	**Firmas**	**Cifrado**	**LWC**	**PQC**	**Gestión flota**
SWUpdate	Sí	Sí	No	No	Externa
hawkBit	Delegada	Delegado	No	No	Sí
Mender	Sí	Sí	No	No	Sí
Balena	Sí	Sí	No	No	Sí
RAUC	Sí	Sí	No	No	Externa

LWC: Lightweight Cryptography; PQC: Post-Quantum Cryptography

Como se observa, ninguna de las soluciones existentes ofrece soporte simultáneo para criptografía ligera y post-cuántica. Esta carencia representa una limitación significativa dado el crecimiento de dispositivos con recursos extremadamente limitados y la amenaza emergente de la computación cuántica. El presente trabajo se construye sobre SWUpdate como agente de actualización en el dispositivo, aprovechando su madurez y robustez probada, pero extendiéndolo gracias a su personalización y facil agregación de modulos con una capa de gestión centralizada que incorpora las familias de algoritmos criptográficos avanzadas ausentes en las soluciones actuales.

Este análisis revela una brecha clara: mientras existen frameworks robustos para gestión de actualizaciones y organismos de estandarización que han definido familias de algoritmos criptográficos avanzadas, ninguna solución integra ambas. La siguiente subsección examina el estado de estas familias de algoritmos criptográficos emergentes.

1.3.3 Necesidad de algoritmos criptográficos eficientes

La enorme variedad de dispositivos IoT, desde chips minúsculos hasta sistemas potentes, hace que necesitemos diferentes tipos de criptografía. Para las actualizaciones OTA, hay dos familias clave: la criptografía ligera para los dispositivos pequeños, y la post-cuántica para protegernos del futuro.

Criptografía ligera: ASCON y el estándar NIST LWC.

El NIST ha elegido ASCON [16] como el estándar para criptografía ligera. Está hecho a medida para dispositivos con pocos recursos y ofrece cifrado autenticado (AEAD), lo que garantiza que los datos son confidenciales y no han sido modificados.

Para las actualizaciones OTA, ASCON es ideal: es hasta cinco veces más eficiente que AES si no tienes hardware dedicado, consume muy poca memoria y protege tanto el firmware como sus metadatos, evitando ataques de repetición o modificación.

Criptografía post-cuántica: preparación ante amenazas futuras.

Los ordenadores cuánticos son una amenaza real a medio plazo para los algoritmos que usamos hoy (RSA, ECDSA). Como los dispositivos IoT duran muchos años, es de sentido común pensar en esto desde ya.

El NIST ha seleccionado tres esquemas de firma post-cuántica: ML-DSA (buen equilibrio), Falcon (firmas pequeñas) y SPHINCS+ (muy seguro teóricamente). Las firmas digitales son clave para asegurar que el firmware es auténtico, incluso frente a ataques cuánticos.

Que el NIST haya estandarizado estos algoritmos significa que técnicamente es posible usarlos. Pero integrarlos en plataformas reales va despacio. En este proyecto he elegido ML-DSA como esquema principal para las firmas digitales, combinándolo con cifrado ligero.

1.4 Antecedentes

Este trabajo tiene su origen en las prácticas realizadas en Ikerlan durante el master, donde se propuso la idea de crear un servicio de distribución de actualizaciones OTA para dispositivos con recursos limitados.

Se llevó a cabo una búsqueda y evaluación de diferentes estrategias para la distribución de actualizaciones en dispositivos IoT. Tras analizar distintas alternativas, se acabó probando una arquitectura basada en la integración de tres componentes: **WFX** como motor de flujos de trabajo para orquestar los estados de la actualización, un **servicio gestor de actualizaciones** encargado de la lógica de distribución en el backend, y **SWUpdate** como agente de actualización instalado en el dispositivo.

El resultado de esa fase inicial fue un prototipo funcional con una interfaz web simple que permitía gestionar actualizaciones de forma básica. Sin embargo, este prototipo presentaba limitaciones significativas:

- **Uso limitado de las funcionalidades:** La interfaz web se diseñó como una prueba de concepto, por lo que no exprimía todo el potencial del gestor de actualizaciones subyacente. Esta limitación se veía agravada por un backend incompleto, que impedía el uso de capacidades avanzadas como los workflows por fases, el rollout progresivo y la monitorización detallada.
- **Backend sin capacidades criptográficas:** El servicio de gestión de actualizaciones carecía completamente de funcionalidades para cifrar o firmar los paquetes de actualización. No existía integración con ningún sistema de gestión de claves para firma o cifrado ni mecanismos para aplicar medidas de seguridad antes de distribuir las actualizaciones, dejando los paquetes expuestos a manipulación o interceptación durante su distribución.
- **Cifrado en dispositivo limitado y sin uso:** Aunque SWUpdate disponía de capacidades criptográficas básicas, estas no se aprovechaban en el prototipo debido a la ausencia de soporte en el backend. Adicionalmente, la configuración de seguridad en el dispositivo era rígida, ofreciendo poca flexibilidad para personalizar algoritmos criptográficos o adaptarse a diferentes requisitos de seguridad según el tipo de dispositivo o escenario de despliegue.
- **Falta de integración con infraestructura PKI:** El servicio estaba concebido para integrarse con Lamassu, una infraestructura de clave pública (PKI) especializada en la gestión de identidad

y ciclo de vida de dispositivos IoT. Sin embargo, esta integración no se había materializado en el prototipo.

Estas limitaciones dieron lugar a la necesidad de un desarrollo más completo y ambicioso, que es precisamente el objetivo de este TFM: partir de esa base probada y construir una plataforma robusta, con soporte criptográfico avanzado y una interfaz de gestión completa que aproveche al máximo las capacidades de cada componente.

1.5 Objetivos

Por tanto, los objetivos de este proyecto nacen directamente de los problemas encontrados en el prototipo anterior. La idea es resolver esas limitaciones (funcionalidades restringidas, criptografía básica y gestión incompleta) desarrollando una plataforma nueva que cubra todas esas carencias.

A partir de esta base y del análisis de la problemática general, se establecen los siguientes objetivos para este Trabajo de Fin de Máster.

1.5.1 Propósito y alcance

Visto el panorama de vulnerabilidades en el ecosistema IoT, las limitaciones de las plataformas OTA existentes y la brecha entre la investigación y la aplicación, este trabajo propone una solución que combina tecnologías consolidadas junto a algoritmos criptográficos modernos.

La estrategia adoptada consiste en utilizar **SWUpdate** como instalador de actualizaciones en el dispositivo, aprovechando su madurez y robustez probada en entornos de producción industrial, pero extendiendo sus capacidades criptográficas para incluir soporte tanto para **algoritmos de criptografía ligera (LWC)** como para **esquemas post-cuánticos**. Esta extensión se integra con una **plataforma de gestión centralizada de flotas de dispositivos**, diseñada para escalar horizontalmente y proporcionar trazabilidad completa del ciclo de vida de las actualizaciones.

Objetivo general:

Diseñar, implementar y validar una plataforma de actualización OTA segura y escalable para dispositivos IoT basada en SWUpdate, extendida con soporte para criptografía ligera (LWC) y post-cuántica, e integrada con una arquitectura de gestión centralizada de flotas, abordando las limitaciones identificadas en soluciones existentes.

Objetivos específicos:

1. **Implementar cifrado de actualizaciones:** Desarrollar capacidades completas de cifrado simétrico en dos niveles: por un lado, implementar en el backend la generación de paquetes cifrados con múltiples algoritmos (AES en sus variantes CBC, CTR y GCM, y ASCON para criptografía ligera); y por otro lado, extender SWUpdate en el dispositivo para descifrar estos paquetes con configuración flexible y adapatable a cada actualizacion con los algoritmos mencionados.

2. **Implementar firma digital de actualizaciones:** Desarrollar mecanismos de firma digital por un lado, implementar en el backend la firma de paquetes con algoritmos tradicionales (RSA, ECDSA) y post-cuánticos (ML-DSA); y por otro lado, extender SWUpdate en el dispositivo con capacidades de verificación de estas firmas, mitigando el riesgo de distribución de firmware malicioso documentado en estudios como «AoT: Attack on Things» [4].

3. **Integración con infraestructura PKI Lamassu:** Integrar el sistema de actualización con Lamassu tanto a nivel de servicio backend como en la interfaz web, aprovechando su infraestructura de gestión de claves, dispositivos, certificados digitales y políticas de seguridad para establecer cadenas de confianza robustas en todo el ecosistema de dispositivos.

4. **Integración continua (CI):** Implementar pipelines de integración continua que ejecuten automáticamente tests unitarios y de extremo a extremo (e2e) del servicio de gestión, validando la correcta interoperabilidad de los componentes y el funcionamiento completo de la arquitectura en cada cambio del código.

5. **Despliegue continuo (CD):** Automatizar la construcción de imágenes de contenedor versionadas de todos los componentes de la plataforma y su despliegue en infraestructura cloud basada en Kubernetes, garantizando escalabilidad horizontal y alta disponibilidad del sistema de gestión.

1.6 Planificación

Para la consecución de los objetivos de este Trabajo de Fin de Máster, se ha definido un plan de trabajo dividido en tres fases principales, que abarcan desde la investigación inicial hasta el despliegue y validación de la plataforma.

1.6.1 Fase 1: Investigación

El objetivo de esta fase es establecer las bases teóricas y técnicas del proyecto, analizando el estado del arte y seleccionando las tecnologías más adecuadas. En esencia, se busca analizar los estándares actuales, identificar qué algoritmos son usados comúnmente y evaluar las nuevas alternativas disponibles, tal como se ha comentado en la introducción.

1.1 Revisión de estándares actuales. Se realizará un estudio exhaustivo de las normativas y recomendaciones de seguridad vigentes para dispositivos IoT (como las guías de NIST y ETSI), así como de los mecanismos de actualización OTA utilizados actualmente en la industria.

1.2 Evaluación de algoritmos lightweight y post-cuánticos. Se analizarán y compararán diferentes algoritmos criptográficos, poniendo el foco en aquellos de la familia de criptografía ligera (LWC) para dispositivos con recursos limitados y en los esquemas de criptografía post-cuántica (PQC) para garantizar la seguridad a largo plazo.

1.3 Elección de algoritmos para los escenarios planteados. Basándose en la evaluación anterior, se seleccionarán los algoritmos de firma y cifrado más adecuados para los distintos escenarios de uso definidos, equilibrando seguridad y rendimiento.

1.6.2 Fase 2: Diseño e integración de módulos de seguridad para dispositivos IoT

Esta fase constituye el núcleo del desarrollo técnico, donde se diseñan e implementan los componentes de la plataforma. Se incluye el desarrollo del frontend, que es la web que permitirá a los desarrolladores hacer uso de toda la infraestructura.

2.1 Diseño de la arquitectura requerida. Definición de la arquitectura global del sistema, especificando los componentes del backend, la base de datos, la API de comunicación y la estructura del cliente en el dispositivo IoT.

2.2 Integración de algoritmos de cifrado y/o firma. Implementación e integración de las librerías criptográficas seleccionadas en la Fase 1 dentro de los servicios de la plataforma, habilitando las capacidades de firma digital y cifrado de los paquetes de actualización tanto en el backend como en el dispositivo IoT.

2.3 Desarrollo del módulo de distribución segura de actualizaciones. Creación del servicio encargado de gestionar el repositorio de actualizaciones y distribución de las actualizaciones de forma segura a los dispositivos autorizados.

2.4 Integración de firma y cifrado Backend y Frontend. Desarrollo de la interfaz web (frontend) y su conexión con el backend, permitiendo a los administradores y desarrolladores subir nuevos firmwares, firmarlos criptográficamente y gestionar las campañas de actualización de manera intuitiva.

2.5 Contenerización, pruebas y despliegue en k8s. Construcción de imágenes de contenedor (Docker) para los microservicios, ejecución de pruebas de integración y despliegue en un clúster de Kubernetes (k8s) para garantizar la escalabilidad y disponibilidad.

1.6.3 Fase 3: Automatización de pruebas, despliegue y ciclo de vida

La fase final se centra en la calidad, la automatización y la validación del sistema en entornos realistas.

3.1 Creación o adaptación del pipeline CICD. Implementación de flujos de CI/CD que ejecutan pruebas automáticamente en cada commit. Además, se habilita una etapa opcional para la construcción de imágenes de contenedor versionadas y su despliegue en Kubernetes.

3.2 Definición de tests unitarios y e2e. Desarrollo de una batería de pruebas que incluye tests unitarios y de extremo a extremo (e2e) para validar la interoperabilidad de los componentes y el flujo completo de actualización.

3.3 Monitorización básica y validación. Supervisión del despliegue y funcionamiento de los servicios mediante el análisis de logs y herramientas de gestión de contenedores, verificando la correcta disponibilidad de los recursos en el clúster.

La planificación temporal de estas fases y tareas se detalla en el diagrama de Gantt de la Figura 1.1.

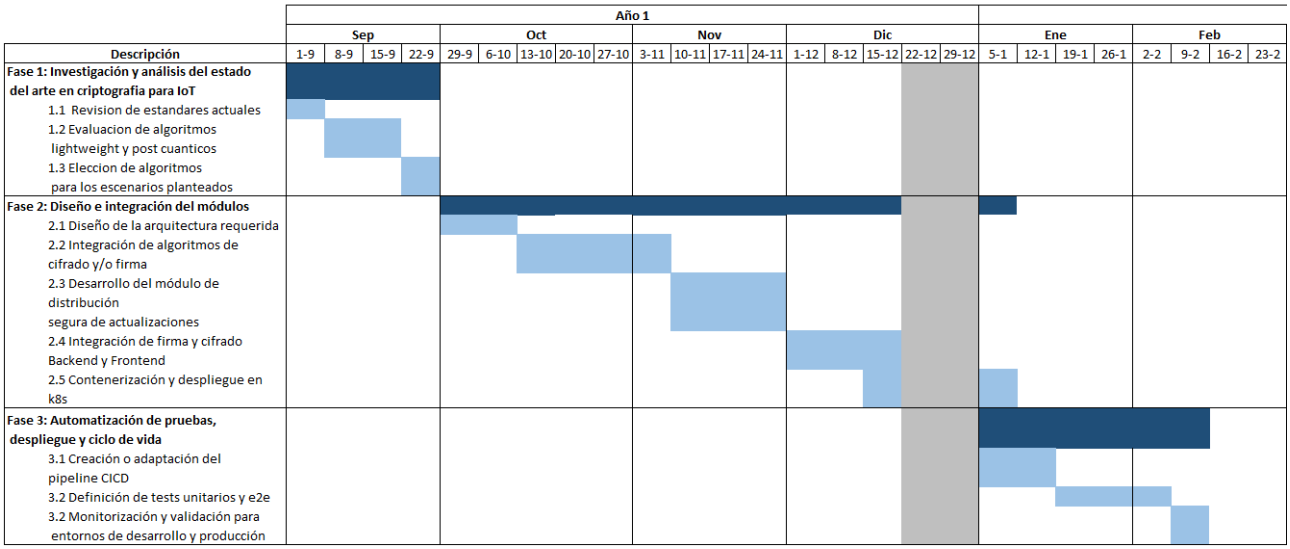

Figura 1.1: *Planificación temporal del proyecto (Diagrama de Gantt)*

1.7 Pliego de condiciones

Este apartado detalla los recursos, requisitos y condiciones específicas necesarias para el desarrollo, despliegue e implantación de la plataforma de actualización OTA segura para dispositivos IoT. El pliego se ha adaptado a las características particulares de este proyecto, siguiendo las directrices de la norma UNE 157001 de AENOR y las especificaciones de la EPS-MU.

1.7.1 Descripción del producto

La plataforma desarrollada en este TFM constituye un sistema distribuido de gestión de actualizaciones OTA compuesto por:

- **Módulo de gestión centralizado (Updates):** Servicio backend implementado en Go que orquesta la creación, firma, cifrado y distribución de paquetes de actualización. Proporciona una API REST para la gestión del ciclo de vida completo de las actualizaciones.
- **Interfaz web de administración:** Aplicación frontend desarrollada que permite a los administradores gestionar flotas de dispositivos, crear campañas de actualización, monitorizar el estado de los despliegues y configurar políticas de seguridad criptográfica.

- **Motor de flujos de trabajo (WFX):** Sistema de orquestación de estados que gestiona el ciclo de vida de las actualizaciones en los dispositivos, soportando workflows directos, por fases y personalizados.
- **Agente de actualización en dispositivo:** Cliente basado en SWUpdate extendido con soporte para algoritmos tradicionales, de criptografía ligera (ASCON) y post-cuántica (ML-DSA), que se ejecuta en los dispositivos IoT con Linux embebido.

El sistema garantiza la confidencialidad mediante cifrado AEAD, la integridad y autenticidad mediante firmas digitales, y proporciona trazabilidad completa del proceso de actualización.

1.7.2 Requisitos de hardware y software para el entorno de desarrollo

Para la configuración del entorno de desarrollo de la plataforma se requieren los siguientes componentes:

1.7.2.1. Hardware mínimo requerido

- **Procesador:** CPU x86_64 de 4 núcleos o superior .
- **Memoria RAM:** 16 GB mínimo (recomendado: 32 GB para desarrollo con contenedores y máquinas virtuales simultáneas).
- **Almacenamiento:** 100 GB de espacio libre en disco SSD para repositorios, contenedores, imágenes de sistema y artefactos de compilación.
- **Red:** Conexión de red estable para descarga de dependencias, acceso a registros de contenedores y comunicación con servicios en la nube.

1.7.2.2. Software de desarrollo requerido

- **Sistema operativo:** Linux (Ubuntu 22.04 LTS o superior, Debian 12, Fedora 38+). Otras distribuciones son compatibles siempre que cumplan los requisitos de las herramientas.
- **Lenguajes y runtimes:**
  - Fork de Go mantenido por Cloudflare (`cfgo`, `github.com/cloudflare/go`) basado en la versión 1.21 o superior (estrictamente requerido para el soporte de criptografía poscuántica en el desarrollo del módulo Updates).
  - Node.js 18 LTS o superior y npm/yarn (para el desarrollo del frontend).
  - Python 3.10+ (para scripts de automatización y SWUGenerator).
  - GCC/G++ 11+ (para compilación de componentes nativos y SWUpdate).

- **Herramientas de contenedores y orquestación:**
  - Docker Engine 24.0+ o Podman 4.0+.
  - Docker Compose 2.0+ para desarrollo local.
  - Kubernetes 1.27+ (minikube, kind o k3s para desarrollo local; clúster productivo para despliegue).
  - kubectl y helm para gestión de despliegues.
- **Librerías criptográficas:**
  - OpenSSL 3.5 o superior (requisito crítico para soporte de algoritmos post-cuánticos).
  - Implementación oficial de ASCON en C (versión 1.3.0 o superior).
- **Herramientas de compilación para sistemas embebidos:**
  - Yocto Project 4.0+ (Kirkstone) para generación de imágenes Linux embebidas.
  - Buildroot 2023.02+ como alternativa para sistemas más restringidos.
  - Toolchains cruzados para las arquitecturas objetivo (ARM, ARM64, RISC-V, etc.).
- **Control de versiones y CI/CD:**
  - Git 2.30+.
  - GitHub Actions para pipelines de integración continua.

1.7.3 Requisitos de hardware y software para dispositivos objetivo

Los dispositivos IoT que ejecutarán el agente de actualización deben cumplir las siguientes especificaciones mínimas:

1.7.3.1. Hardware del dispositivo

- **Procesador:** ARM Cortex-A5 o superior, o equivalente (x86, RISC-V).
- **Memoria RAM:** 16 MB mínimo.
- **Almacenamiento:**
  - Flash/eMMC/SD con capacidad para esquema A/B: mínimo 2× tamaño del rootfs + 20 % de margen.
  - Ejemplo: para rootfs de 200 MB, se requieren al menos 500 MB de almacenamiento total.
- **Conectividad:** Ethernet, Wi-Fi o módulo (4G/5G) para descarga de actualizaciones OTA.

1.7.3.2. Software del dispositivo

- **Sistema operativo:** Linux embebido (kernel 5.10 LTS o superior) generado mediante Yocto Project o Buildroot.
- **Bootloader:** U-Boot 2022.01+ con soporte para variables de entorno persistentes y arranque condicional para estrategia A/B.
- **Estructura de particiones:** Esquema de doble partición (Slot A / Slot B) para rootfs y kernel, con partición de datos persistente separada.
- **SWUpdate:** Versión modificada con soporte para ASCON y ML-DSA (proporcionada como parte de este proyecto).
- **Librerías criptográficas en dispositivo:**
  - OpenSSL 3.5+ (compilado estáticamente o como librería compartida).
  - Implementación de ASCON integrada en SWUpdate.
- **Agente SWUpdate configurado:** Cliente compilado con módulo WFX habilitado para comunicación con el backend de gestión.

1.7.4 Requisitos del entorno de producción

El despliegue en producción de la plataforma de gestión centralizada requiere:

1.7.4.1. Infraestructura de nube o on-premise

- **Clúster Kubernetes:**
  - Versión 1.24+ (compatible con las versiones de Helm y componentes utilizados)
  - Al menos 1 nodo worker (recomendado 3 para alta disponibilidad)
- **Recursos por nodo:**
  - CPU: 4 núcleos (2.0 GHz o superior)
  - RAM: 8 GB
  - Almacenamiento: 50 GB (SSD recomendado)
- **Balanceador de carga:**
  - LoadBalancer service support (para exposición externa)
  - Ingress controller (se incluye Envoy Gateway en el chart)
- **Almacenamiento de objetos:**

- StorageClass con soporte ReadWriteOnce (para PVCs persistentes)
- Mínimo 13 GB distribuidos: Updates (11GB), VA (1GB), KMS (1GB)

- **Base de datos:**
  - PostgreSQL 13+ (incluido en el despliegue)

1.7.4.2. Requisitos de red

- **Ancho de banda:** Suficiente para soportar descargas simultáneas de firmware por los dispositivos de la flota (se recomienda planificar despliegues por lotes).
- **Certificados SSL/TLS:** Certificados válidos para comunicación cifrada entre dispositivos y backend (Let's Encrypt, CA corporativa).
- **Conectividad dispositivos:** Los dispositivos deben poder alcanzar la API del backend (puertos HTTPS 443) mediante resolución DNS o IP estática.

1.7.5 Condiciones de seguridad

La plataforma debe operar bajo las siguientes condiciones de seguridad:

- **Gestión de claves criptográficas:**
  - Las claves privadas de firma deben almacenarse en un HSM (Hardware Security Module) o mediante PKCS#11.
  - Las claves simétricas de cifrado deben gestionarse a través del módulo KMS.
  - Rotación de claves conforme a política definida (recomendado: cada 12 meses o tras compromiso detectado).
- **Autenticación y autorización:**
  - Control de acceso basado en roles (RBAC) para administradores de la plataforma.
- **Integridad y trazabilidad:**
  - Las actualizaciones pueden estar firmada digitalmente (ECDSA, RSA o ML-DSA según configuración).
  - Registro auditable de todas las operaciones: creación de paquetes, despliegues, éxitos y fallos.

1.7.6 Condiciones de cumplimiento normativo

El sistema ha sido diseñado para facilitar el cumplimiento de las siguientes normativas:

- **Cyber Resilience Act (CRA):** Requisitos de actualización segura durante todo el ciclo de vida del producto, reporte de vulnerabilidades y gestión de incidentes.
- **IEC 62443-4-2 (CR 3.10 - Support for Updates):** Mecanismos robustos para la actualización de componentes de software y firmware en sistemas de control industrial.
- **NIST SP 800-193:** Protección de la integridad de la plataforma mediante actualizaciones autenticadas y rollback seguro.
- **ETSI EN 303 645:** Requisitos de ciberseguridad para dispositivos IoT de consumo, incluyendo actualizaciones seguras y gestión del ciclo de vida.

1.7.7 Condiciones de implantación para cliente final

En proyectos donde se despliegue esta plataforma para un cliente específico, se deben considerar las siguientes condiciones adicionales:

1.7.7.1. Plan de onboarding de dispositivos

- **Provisión inicial:** Configuración de certificados digitales para cada dispositivo.
- **Registro en DMS:** Asignación de dispositivos a grupos lógicos (Device Management Systems) según criterios del cliente (geográficos, funcionales, etc.).

1.7.7.2. Soporte y mantenimiento

- Definición de SLA (Service Level Agreement) para disponibilidad del servicio de actualizaciones.
- Política de actualizaciones de seguridad de la propia plataforma (parches del backend, dependencias críticas).
- Monitorización continua y alertas ante fallos de actualización en dispositivos.

1.7.8 Condiciones de ejecución del proyecto

Durante el desarrollo de este TFM se han aplicado las siguientes condiciones:

- **Metodología ágil:** Desarrollo iterativo con entregas incrementales y revisiones periódicas con tutor.

- **Control de versiones:** Todo el código fuente se gestiona mediante Git con repositorios públicos para componentes open source y privados para extensiones propietarias.
- **Pruebas continuas:** Pipeline CI/CD que ejecuta tests unitarios y de integración en cada commit.

2. Desarrollo

En esta sección se desarrollan los aspectos técnicos y conceptuales que sustentan la plataforma propuesta. En primer lugar se tratarán las bases teóricas de la criptografía simétrica y asimétrica, necesarias para comprender las decisiones de diseño posteriores. A continuación, se explicará la terminología clave y las herramientas empleadas en los flujos de actualización de los dispositivos. Finalmente, se definirá la plataforma de desarrollo propuesta y su modelo de datos.

2.1 Bases teóricas de la criptografía

En el mundo de la seguridad digital, la criptografía desempeña un papel fundamental en el cifrado y descifrado de información. Su objetivo principal es asegurar la confidencialidad e integridad de los datos, protegiéndolos contra accesos no autorizados durante la transmisión o almacenamiento. Esta sección explora los fundamentos de la criptografía, con un énfasis particular en dos formas principales: la criptografía simétrica y asimétrica.

En la criptografía, existen varios términos clave que son fundamentales para entender cómo funciona este campo. A continuación, se definen algunos de los conceptos más básicos:

- **Mensaje**: Es la información original que se desea proteger.
- **Criptograma**: Es el resultado del proceso de cifrado, que oculta el mensaje original utilizando algún algoritmo criptográfico.
- **Criptoanálisis**: Es el estudio de los sistemas criptográficos con el fin de encontrar debilidades que permitan recuperar el mensaje original sin conocer la clave de cifrado.

Por otro lado, existen principios clave que son esenciales para proteger la información y los sistemas informáticos de accesos no autorizados, alteraciones y destrucción. Estos principios son la base de cualquier estrategia de seguridad y son fundamentales para garantizar la protección de los datos. A continuación, se presentan brevemente estos principios:

- **Confidencialidad:** Este principio asegura que la información es accesible solo para aquellas personas autorizadas a tener acceso. Protege los datos de accesos no autorizados y garantiza la privacidad de la información.

- **Integridad:** La integridad se refiere a la exactitud y completitud de la información y los métodos de procesamiento. Este principio asegura que la información no ha sido alterada de manera no autorizada y que se mantiene tal como fue creada, enviada y recibida.
- **No repudio:** Garantiza que una vez realizada una transacción, no se pueda negar su autoría, proporcionando evidencia de la participación de las partes involucradas.
- **Autenticidad:** La autenticidad garantiza que las transacciones, las comunicaciones y los datos son genuinos y que las identidades de las partes involucradas son verificadas. Este principio evita la suplantación de identidad y asegura que los datos provienen de una fuente legítima.

2.1.1 Criptografía simétrica

La criptografía simétrica, también conocida como criptografía de clave secreta, es un método de cifrado donde se utiliza la misma clave tanto para cifrar como para descifrar la información. Esta clave debe ser compartida entre el emisor y el receptor de manera segura antes de iniciar la comunicación.

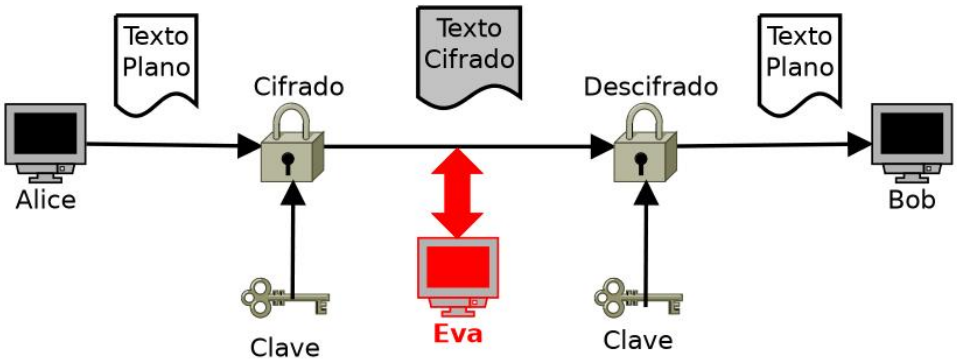

Figura 2.1: *Uso de cifrado simétrico*

Un ejemplo clásico de cifrado simétrico es el cifrado César, que consiste en desplazar cada letra del mensaje n posiciones en el alfabeto:

$$E(x) = (x + n) \bmod 26$$

Asumiendo que n es 1 y x la posición de la letra en el abecedario, la palabra “Hola” pasaría a ser “Ipmb”. En este caso, la clave sería el número de posiciones a desplazar.

La criptografía simétrica presenta ciertas limitaciones importantes:

- **Distribución de claves:** La clave debe ser transmitida de manera segura entre ambas partes. Si

la clave es interceptada por un tercero, toda la comunicación cifrada con esa clave se vuelve insegura.

- **Escalabilidad:** A medida que el número de participantes aumenta, el manejo de claves se vuelve más complejo, ya que cada par de usuarios necesita una clave única y segura.

Desde la perspectiva de la computación cuántica, el cifrado simétrico goza de una posición relativamente privilegiada. Según el NIST, algoritmos simétricos como AES-256 siguen siendo seguros a largo plazo incluso ante amenazas de computación cuántica [16]. Esto se debe a que el algoritmo de Grover, que es el ataque cuántico más efectivo contra criptografía simétrica, reduce la seguridad efectiva a la mitad del tamaño de clave. Por ejemplo, una clave de 256 bits proporciona una seguridad equivalente a 128 bits ante un ataque cuántico, lo que sigue siendo considerado seguro por el NIST para aplicaciones sensibles.

2.1.1.1. Autenticación usando cifrado simétrico: AEAD

Para superar algunas de las limitaciones del cifrado simétrico básico y proporcionar mayor seguridad, se utilizan modos de cifrado avanzados como AEAD (Authenticated Encryption with Associated Data). AEAD combina el cifrado con la autenticación en una sola operación, garantizando tanto la confidencialidad como la integridad de los datos.

AEAD opera con tres componentes principales:

- **Clave secreta:** Compartida entre emisor y receptor.
- **Datos a cifrar (plaintext):** El mensaje original.
- **Datos asociados (associated data):** Información adicional que debe autenticarse pero no cifrarse, como encabezados de protocolo.

El proceso de AEAD produce dos salidas:

- **Criptograma:** Los datos cifrados.
- **Tag de autenticación:** Un código que verifica la integridad de los datos y los datos asociados.

Durante el descifrado, el receptor verifica el tag de autenticación. Si coincide, los datos son válidos y no han sido alterados; de lo contrario, se rechaza el mensaje. Esto previene ataques como la manipulación de datos o la repetición de mensajes.

Un ejemplo de algoritmo AEAD ampliamente utilizado es AES-GCM (Advanced Encryption Standard - Galois/Counter Mode), que combina AES para el cifrado con el modo GCM para la autenticación. Este enfoque es fundamental en protocolos seguros como TLS para proteger las comunicaciones en internet.

2.1.2 Criptografía asimétrica

La criptografía asimétrica introduce un enfoque diferente mediante el uso de un par de claves: una pública y una privada. La clave pública, como su nombre indica, se puede compartir abiertamente, mientras que la clave privada se mantiene en secreto por el usuario.

La característica fundamental es que lo que es cifrado utilizando la clave pública solo puede ser descifrado por su correspondiente clave privada, y viceversa. Esta relación matemática permite establecer comunicaciones seguras sin necesidad de compartir una clave secreta previamente.

- **Cifrado**: $C = E_{k_{\text{publica}}}(M)$
- **Descifrado**: $M = D_{k_{\text{privada}}}(C)$

La criptografía de clave pública y privada se puede entender mediante el símil de una caja con dos cerraduras, cada una correspondiente a una llave. Inicialmente, cualquiera de las dos llaves puede cerrar la caja, pero para abrirla se debe utilizar la llave opuesta. Imaginemos el siguiente escenario con Alice y Bob. Para este ejemplo, Bob tiene dos tipos de llaves: una pública y otra privada. La clave pública la hará disponible para todos, haciendo copias de ella y dándoselas a cualquiera que la pida. La clave privada, en cambio, será mantenida en secreto y solo Bob dispondrá de ella (ver Figura 2.2).

Figura 2.2: *Creación de las llaves por parte de Bob*

2.1.2.1. Cifrado con clave pública (Confidencialidad)

1. **Cerrando la caja (Encriptación con la clave pública):** Alice quiere enviar un mensaje confidencial a Bob. Para hacerlo, primero coloca el mensaje en la caja, y posteriormente utiliza la llave pública de Bob para cerrarla. Al cerrar la caja con la llave pública de Bob, Alice garantiza que solo Bob podrá abrirla, dado que solo él posee la llave privada correspondiente (ver Figura 2.3).

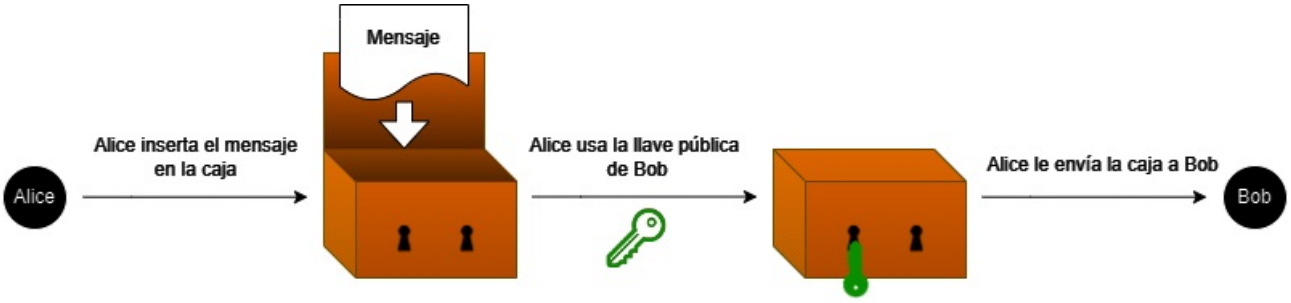

Figura 2.3: *Alice introduce el mensaje en la caja y la cierra con la clave pública de Bob.*

2. **Abriendo la caja (Descifrado con la clave privada):** Cuando Bob recibe la caja cerrada, utiliza su llave privada para abrirla. Esta llave privada es única y está en su exclusiva posesión. Nadie más puede abrir la caja porque no disponen de la llave privada de Bob (ver Figura 2.4).

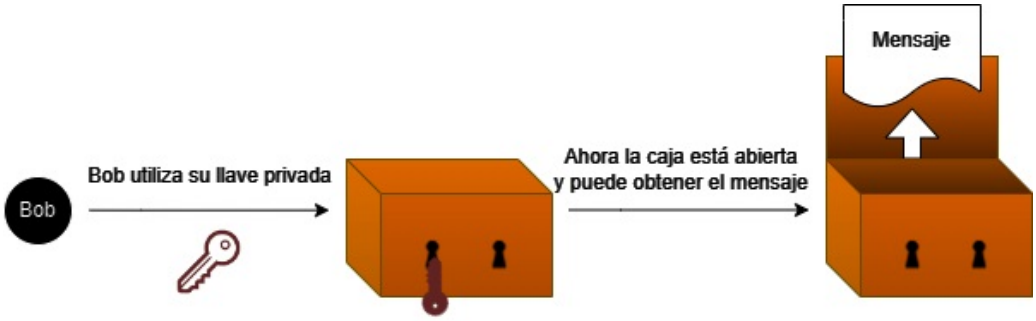

Figura 2.4: *Bob abre la caja y obtiene el mensaje.*

2.1.2.2. Cifrado con clave privada (Autenticidad)

En el ejemplo anterior se ha demostrado cómo la criptografía asimétrica puede garantizar la confidencialidad. Además, puede garantizar la autenticidad. Para ello, veamos el siguiente ejemplo donde Bob quiere enviarle un mensaje a Alice y demostrar que es Bob realmente el que lo ha escrito.

1. **Cerrando la caja (Encriptación con la llave privada):** Bob coloca el mensaje en la caja y la cierra utilizando su llave privada (ver Figura 2.5). Esto asegura que cualquier persona con acceso a su llave pública puede abrir la caja y leer el mensaje, pero garantiza que solo Bob pudo haberla cerrado, ya que solo él posee la llave privada.

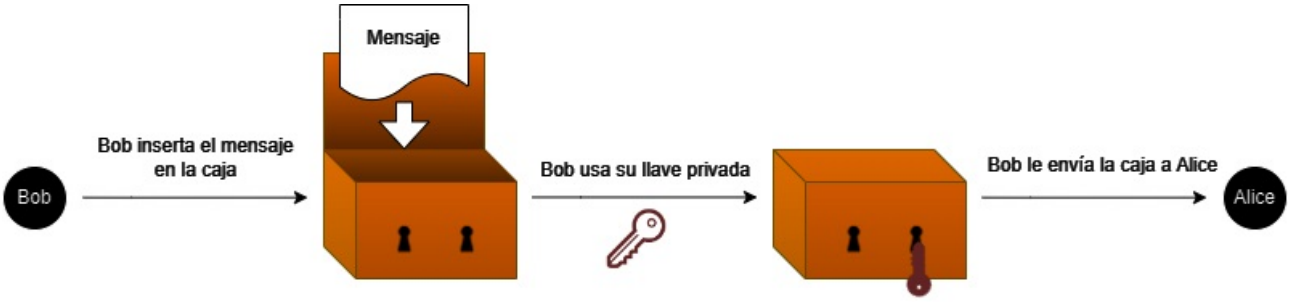

Figura 2.5: *Bob inserta el mensaje y usa su llave privada.*

2. **Abriendo la caja (Descifrado con la clave pública):** Alice recibe la caja y utiliza la clave pública de Bob para abrirla (ver Figura 2.6). Al hacerlo, Alice puede estar segura de que el mensaje fue realmente enviado por Bob, ya que solo la llave pública de Bob puede abrir la caja que fue cerrada con su llave privada. Esto proporciona autenticidad y no repudio, ya que Bob no puede negar haber enviado el mensaje.

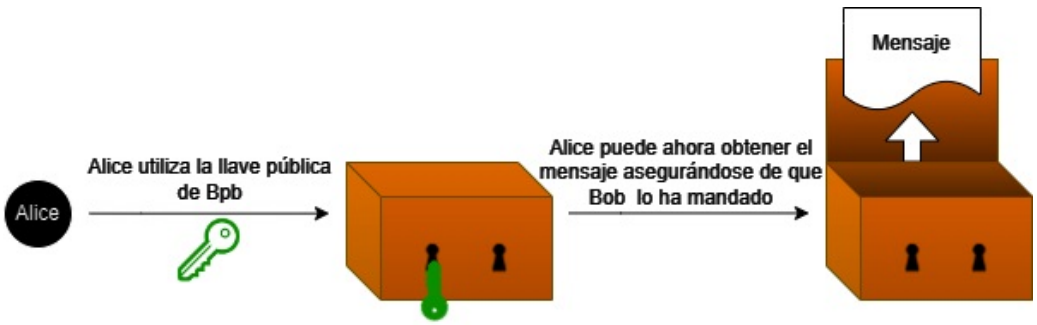

Figura 2.6: *Alice utiliza la clave pública de Bob para abrir la caja y verificar la autenticidad del mensaje.*

Mediante estos dos mecanismos, la criptografía asimétrica ofrece no solo confidencialidad, sino también no repudio y autenticidad.

2.1.3 Principios de seguridad asegurados con cifrado asimétrico

Con el cifrado asimétrico únicamente se logran los siguientes principios:

- **Confidencialidad:** Se cumple, ya que el cifrado asimétrico permite proteger los datos para que solo sean accesibles por las personas autorizadas con la clave privada correspondiente.
- **Integridad:** No se puede asegurar completamente, ya que no hay un mecanismo inherente en el cifrado asimétrico que garantice que los datos no han sido alterados.
- **No repudio:** No se puede garantizar completamente, ya que el cifrado asimétrico por sí solo no proporciona una manera de evitar que una de las partes niegue haber realizado una acción.
- **Autenticidad:** No se puede asegurar completamente, ya que el cifrado asimétrico por sí solo no proporciona un método para verificar la identidad de manera confiable en una comunicación no física.

Es decir, solo podemos verificar la confidencialidad de los datos. En los siguientes apartados se verán los métodos existentes para cumplir el resto de principios.

2.1.4 Firma digital

Una firma digital es un mecanismo de seguridad electrónica que imita la funcionalidad de una firma manuscrita en el entorno digital. Utiliza la criptografía asimétrica.

Al firmar digitalmente un documento o mensaje, se utiliza la clave privada para generar un código único (la firma) basado en el contenido del mensaje mediante una función hash[1]. Este código se adjunta después al mensaje o documento. Cuando el receptor obtiene el mensaje firmado, puede usar la clave pública del remitente para verificar si la firma es válida.

Para crear la firma se siguen los siguientes pasos:

- **Creación del hash**: Primero, el remitente crea un resumen del mensaje original utilizando una función hash criptográfica.
- **Firmado del hash**: A continuación, el remitente cifra el hash del mensaje con su clave privada. La firma digital no es el cifrado completo del mensaje, sino el cifrado del hash del mensaje.
- **Adjuntar al mensaje**: El hash cifrado (la firma) se adjunta al mensaje original. Este mensaje firmado se puede enviar al receptor.

Y para verificar la firma:

- **Separar la firma y el mensaje**.
- **Descifrar la firma**: El receptor utiliza la clave pública del remitente para descifrar la firma, obteniendo así el hash del mensaje que el remitente calculó.
- **Calcular el hash del mensaje**: El receptor calcula el hash del mensaje recibido utilizando la misma función hash que usó el remitente.
- **Comparar los hashes**: El receptor compara el hash que acaba de calcular con el hash descifrado de la firma. Si ambos hashes coinciden, la firma es válida; esto significa que el mensaje no ha sido alterado y que el remitente es quien dice ser.

De esta manera se puede garantizar la autenticidad, la integridad y el no repudio:

- **Integridad**: Si la verificación es exitosa, se puede estar seguro de que el mensaje no ha sido alterado desde que fue firmado.
- **Autenticidad**: Al verificar con éxito la firma digital, se confirma que el mensaje fue realmente enviado por quien posee la clave privada correspondiente.
- **No Repudio**: Una vez enviado un mensaje firmado digitalmente, el remitente no puede negar haberlo enviado, ya que la firma digital creada con su clave privada es única y verificable por cualquier persona que tenga su clave pública.

Al combinar esto con el cifrado asimétrico convencional, también se garantiza la confidencialidad. Sin embargo, la autenticidad completa solo se puede lograr con la introducción de autoridades certificadoras (CA), que se discutirá a continuación.

[1]Una función hash convierte datos de cualquier tamaño en una cadena de texto de tamaño fijo, conocida como hash. Cada conjunto único de datos produce un hash único, y cualquier cambio en los datos cambia el hash.

2.1.5 Autoridades Certificadoras y certificados digitales

En las comunicaciones digitales, es crucial garantizar que las partes involucradas sean quienes dicen ser. Aquí es donde entra en juego la Autoridad Certificadora (CA). Una CA es una entidad de confianza que emite certificados digitales, los cuales verifican la identidad de los individuos, servidores u otras entidades. Sin una CA, sería muy difícil establecer una relación de confianza en entornos donde las partes no se conocen previamente.

2.1.5.1. Autoridad Certificadora

Una Autoridad Certificadora (CA) es una organización responsable de emitir y gestionar certificados digitales. La CA actúa como un tercero de confianza en el ecosistema de la infraestructura de clave pública (PKI). Su papel principal incluye:

- Verificar la identidad de las entidades que solicitan un certificado digital.
- Emitir certificados digitales que enlazan una clave pública con la identidad del solicitante.
- Revocar certificados digitales si ya no son válidos o se comprometen.
- Publicar listas de certificados revocados (CRL).

2.1.5.2. Certificado digital

Un certificado digital es un documento electrónico que utiliza una firma digital para enlazar una clave pública con la identidad del propietario del certificado. Esto se consigue firmando la clave pública del propietario del certificado con la clave privada de la autoridad certificadora. Un certificado digital contiene varios campos clave:

- **Nombre del propietario**: Identifica al propietario del certificado.
- **Clave pública del propietario**: La clave pública asociada con el propietario.
- **Nombre de la CA**: La entidad que emitió el certificado.
- **Firma digital de la CA**: La firma digital de la CA que valida el certificado.
- **Período de validez**: Las fechas de inicio y expiración del certificado.
- **Número de serie del certificado**: Un número único para identificar el certificado.

Los certificados digitales siguen el estándar X.509, que establece el formato y las reglas para su emisión y procesamiento en sistemas de infraestructura de clave pública (PKI).

2.1.5.3. Proceso de Verificación de Certificados

1. **Firma del Certificado**: La CA utiliza su clave privada para firmar el certificado digital del propietario.
2. **Verificación del Certificado**: Cualquier entidad que reciba el certificado puede usar la clave pública de la CA (generalmente disponible y conocida) para verificar la firma del certificado. Si la verificación es exitosa, se confía en la identidad del propietario del certificado.

2.1.5.4. Jerarquía de las Autoridades Certificadoras

Las CA se organizan en una jerarquía, donde las CA de nivel superior, la CA raíz (root CA), firman los certificados de las CA subordinadas (CA intermedio), y estas, a su vez, pueden firmar certificados para usuarios finales o servidores. Esto crea una cadena de confianza que puede ser verificada siguiendo el camino desde un certificado de usuario final hasta la raíz CA.

En la Figura 2.7, se presenta un diagrama que ilustra esta jerarquía:

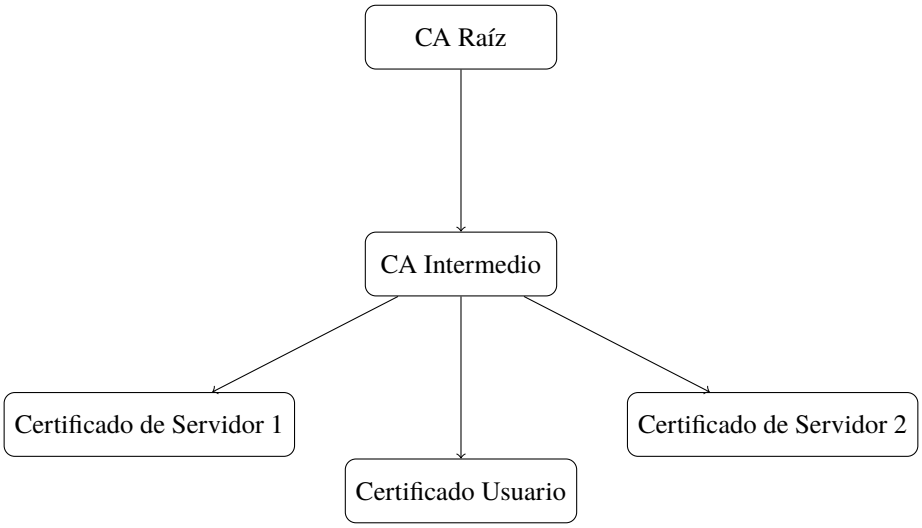

Figura 2.7: *Jerarquía de las Autoridades Certificadoras.*

2.1.6 Estándar de sintaxis de mensajes criptográficos (CMS)

El Cryptographic Message Syntax (CMS), estandarizado en la RFC 5652, es un marco flexible para proteger mensajes. Este estándar define una sintaxis para encapsular datos firmados, cifrados, autenticados o comprimidos.

Una de las características más potentes de CMS es su capacidad para soportar arquitecturas de confianza jerárquicas mediante la inclusión de certificados. En un esquema de firma tradicional, el receptor necesita tener previamente la clave pública del remitente. CMS permite un modelo más dinámico:

1. **Confianza anclada en la CA:** El verificador posee únicamente el certificado de una Autoridad de Certificación (CA) de confianza.

2. **Estructura de datos firmados (SignedData):** El mensaje CMS encapsula no solo el contenido y la firma digital, sino también el certificado digital del firmante (y potencialmente toda la cadena de certificación hasta la CA raíz).

3. **Proceso de validación:** Al recibir el mensaje, el verificador realiza una validación en dos etapas:

   - Primero, extrae el certificado del firmante incluido en el mensaje y comprueba que es válido y que ha sido emitido por la CA de confianza que ya posee.
   - Una vez validada la identidad del firmante, utiliza la clave pública contenida en ese certificado para verificar la firma digital del mensaje.

Este mecanismo desacopla la verificación de la identidad específica del firmante. Permite que el emisor rote sus claves o que existan múltiples entidades emisoras diferentes, siempre que todas posean certificados válidos emitidos por la CA común, el receptor podrá validar los mensajes sin necesidad de reconfiguración.

2.2 Elección de algoritmos criptográficos

Para este proyecto, la selección de algoritmos criptográficos se ha basado en las necesidades actuales del ecosistema IoT y en la gran diversidad de dispositivos que lo componen. El panorama IoT abarca desde dispositivos con recursos extremadamente limitados (pocos kilobytes de RAM y MHz de procesamiento) hasta dispositivos más potentes como gateways o sistemas embebidos basados en Linux. Esta heterogeneidad hace inviable la adopción de una única solución criptográfica que se adapte óptimamente a todos los escenarios.

Actualmente, el estándar de facto para el cifrado simétrico es AES (Advanced Encryption Standard). Este algoritmo es extremadamente robusto y eficiente en plataformas que disponen de recursos suficientes, especialmente aquellas que cuentan con aceleración por hardware (instrucciones AES-NI o similares). Sin embargo, en dispositivos restringidos sin soporte hardware específico, la implementación de AES puede resultar costosa en términos de ciclos de reloj, consumo de energía y uso de memoria, además de ser susceptible a ataques de canal lateral si no se implementa con complejas contramedidas.

En el ámbito de la criptografía asimétrica, y específicamente en lo referente a la firma digital, los estándares actuales más utilizados son RSA y ECDSA. Este último, basado en curvas elípticas, destaca por su eficiencia frente a RSA, permitiendo longitudes de clave mucho menores para un mismo nivel de seguridad. Adicionalmente, en este proyecto se tiene la intención de añadir soporte para ML-DSA (Module-Lattice-Based Digital Signature Standard), un algoritmo de firma post-cuántica que será detallado más adelante.

Por esta razón, se ha decidido ofrecer soporte para múltiples familias de algoritmos, permitiendo que cada despliegue seleccione la opción más adecuada según las capacidades del hardware objetivo.

2.2.1 ASCON: Estándar de criptografía ligera

ASCON ha sido seleccionado por el NIST (National Institute of Standards and Technology) como el estándar oficial de criptografía ligera para dispositivos con recursos limitados, publicado en la especificación NIST SP 800-232 [17]. Esta selección consiste en un proceso de evaluación de varios años en el que ASCON demostró un equilibrio óptimo entre seguridad, eficiencia y versatilidad.

ASCON es una familia de algoritmos que proporciona:

- **Cifrado autenticado (AEAD):** Garantiza confidencialidad e integridad en una sola operación.
- **Funciones hash:** Para verificación de integridad y generación de resúmenes criptográficos.
- **Autenticación de mensajes (MAC):** Para verificar la autenticidad sin cifrado.

Ascon cuenta con 3 variantes principales:

- **Ascon-128:** La variante estándar que ofrece 128 bits de seguridad simétrica. Ideal para la mayoría de las aplicaciones IoT.
- **Ascon-128a:** Versión alternativa de alto rendimiento, que sacrifica ligeramente el tamaño del código por una mayor velocidad de procesamiento.
- **Ascon-80pq:** Variante de seguridad post-cuántica (post-quantum), diseñada con una clave de 160 bits para abordar desafíos de seguridad futuros.

2.2.1.1. Ventajas de rendimiento frente a AES

Según los benchmarks realizados por los autores de ASCON en el documento *Status Update on Ascon v1.2* [18], el algoritmo presenta ventajas significativas en dispositivos sin aceleración hardware para AES:

- **Velocidad:** ASCON es entre 2 y 5 veces más rápido que AES-GCM en implementaciones software sobre microcontroladores de 8, 16 y 32 bits sin instrucciones AES dedicadas.
- **Tamaño de código:** La implementación de ASCON requiere significativamente menos memoria de programa que AES, lo cual es crítico en dispositivos con flash limitada.
- **Uso de RAM:** El estado interno de ASCON (320 bits) es compacto, reduciendo el consumo de memoria durante las operaciones criptográficas.
- **Tamaño de clave:** ASCON opera con claves de 128 bits, ofreciendo un nivel de seguridad adecuado teniendo en cuenta el objetivo de este, la criptografía ligera.

2.2.1.2. Validación experimental

Para validar la elección de ASCON, se decidió realizar pruebas experimentales en dos dispositivos representativos de entornos IoT con recursos limitados y sin aceleración por hardware para criptografía: una Raspberry Pi 3 [19] y un Arduino UNO R4 [20].

La implementación utilizada para las pruebas es la oficial de ASCON (versión 1.3.0) disponible en GitHub [21], la cual es la recomendada por el NIST [22].

Las pruebas en la Raspberry Pi 3 han consistido en el cifrado y descifrado de un fichero de 100 MB. Para asegurar la fiabilidad de los datos y reducir la variabilidad, se ejecutaron 100 iteraciones de cada prueba.

En el caso del Arduino, debido a las limitaciones inherentes de un microcontrolador, las pruebas fueron más restringidas. Se procedió probando con tamaños de texto plano (*plaintext*) incrementales, desde pocos bytes hasta el límite permitido por la memoria del dispositivo.

Los resultados obtenidos para la Raspberry Pi 3 se muestran en la Figura 2.8.

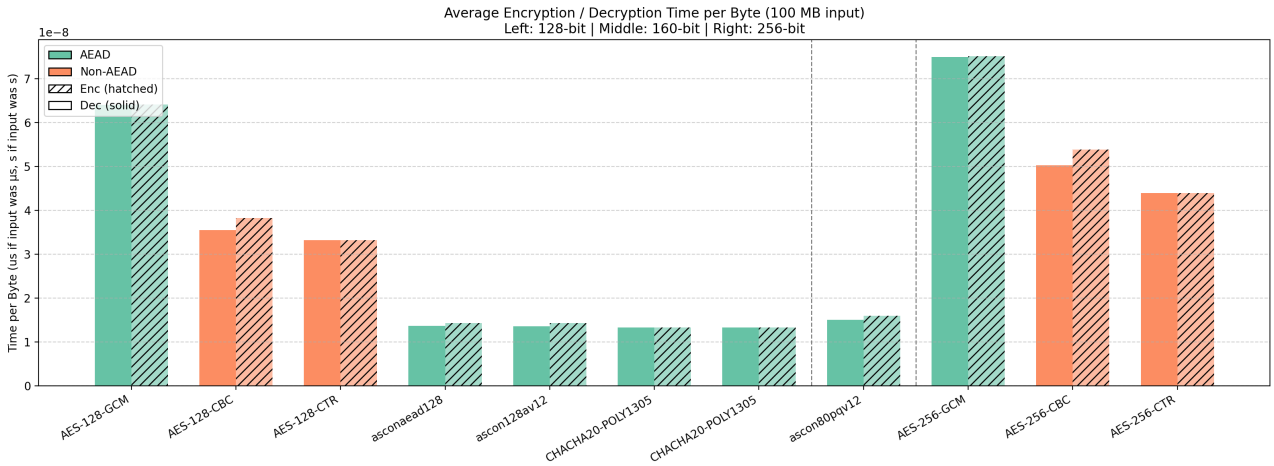

Figura 2.8: *Resultados de rendimiento en Raspberry Pi 3*

Como se puede observar, ASCON confirma su superioridad frente a las variantes de AES, siendo entre 2 y 5 veces más rápido en este entorno sin aceleración hardware. Además, su rendimiento es equiparable al de ChaCha20-Poly1305, otro algoritmo conocido por su eficiencia en software.

Por otro lado, en el entorno más restringido del Arduino, los resultados son mucho más favorables para ASCON en todos los sentidos, como se aprecia en la Figura 2.9.

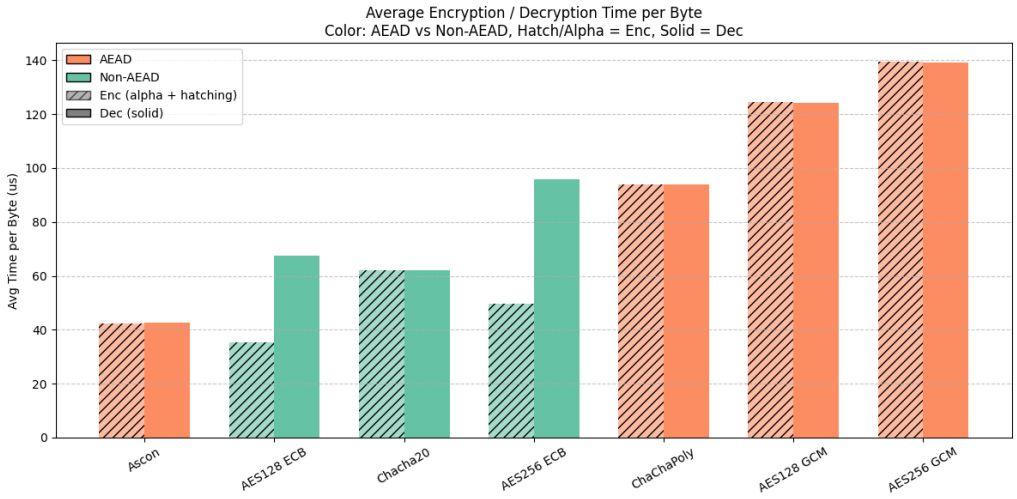

Figura 2.9: *Resultados de rendimiento en Arduino UNO R4*

ASCON resulta ser el claro ganador. Además, es importante destacar la eficiencia en el uso de memoria. En la Figura 2.10 se muestra cómo ASCON permite cifrar y descifrar bloques de texto plano más grandes que el resto de algoritmos antes de agotar los recursos del microcontrolador, lo cual se atribuye a su menor consumo de memoria RAM.

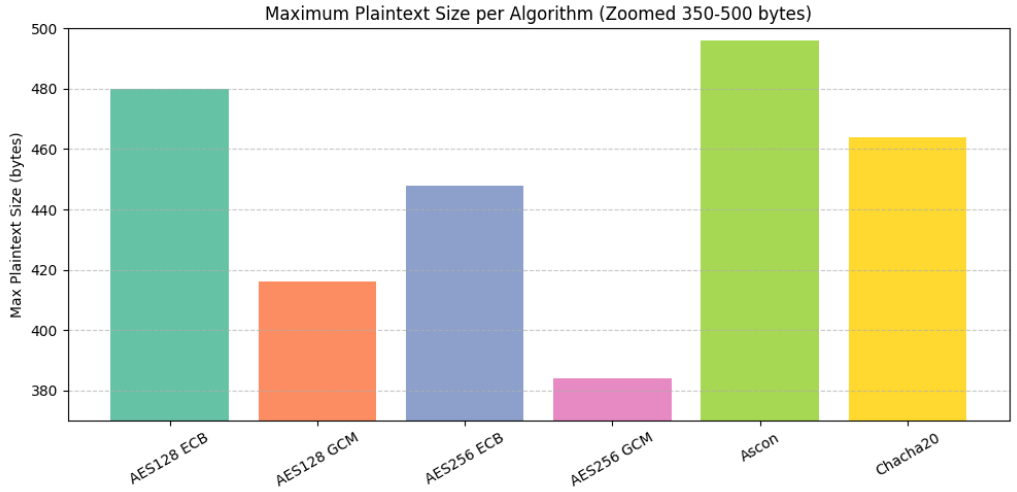

Figura 2.10: *Tamaño máximo de texto plano admitido en Arduino*

Adicionalmente, al considerar el uso de ASCON como algoritmo AEAD, el tamaño del tag de autenticación y los bytes adicionales que se añaden al texto plano son aspectos cruciales a evaluar. En la Figura 2.11 se ilustra la cantidad de bytes extra requeridos por cada algoritmo en comparación con el texto plano original. ASCON ocupa la mejor posición con solo 16 bytes adicionales, junto con AES-CTR y ChaCha20 sin autenticación.

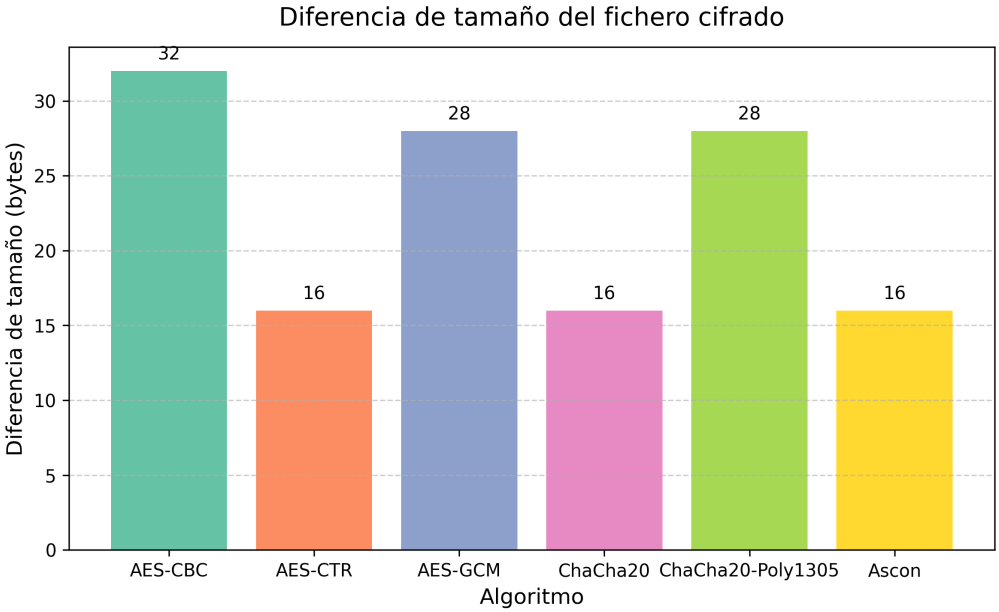

Figura 2.11: *Bytes adicionales requeridos por algoritmos AEAD*

2.2.1.3. Consideraciones de seguridad post-cuántica

Más allá de las ventajas de rendimiento, ASCON ofrece también protección contra amenazas futuras derivadas del desarrollo de computadores cuánticos. Según el NIST, algoritmos simétricos como AES-256 siguen siendo seguros a largo plazo incluso ante amenazas de computación cuántica [16]. Esto se debe a que el algoritmo de Grover, que es el ataque cuántico más efectivo contra criptografía simétrica, reduce la seguridad efectiva a la mitad del tamaño de clave. Por ejemplo, una clave de 256 bits proporciona una seguridad equivalente a 128 bits ante un ataque cuántico, lo que sigue siendo considerado seguro por el NIST para aplicaciones sensibles.

Sin embargo, ASCON-80pq proporciona una protección algo más robusta. Esta variante está específicamente diseñada con una clave de 160 bits para garantizar 80 bits de seguridad efectiva incluso ante ataques cuánticos basados en el algoritmo de Grover [16]. Si bien no es la variante más recomendada para entornos con recursos abundantes, donde AES-256 ofrecería una mayor seguridad a largo plazo, no está mal considerada, especialmente en dispositivos con recursos limitados o en entornos IoT donde la eficiencia computacional es un factor crítico. Esta característica lo hace particularmente atractivo para entornos IoT donde tanto la eficiencia computacional como la resistencia cuántica a largo plazo son requisitos críticos.

2.2.1.4. Seguridad post cuantica en la firma digital: ML-DSA

El panorama es significativamente diferente en el caso de la criptografía asimétrica. El algoritmo de Shor [23], propuesto en 1997, demostró teóricamente que los sistemas criptográficos asimétricos actuales, como RSA y ECDSA, son completamente vulnerables a un computador cuántico suficientemente potente. Este algoritmo puede factorizar números grandes y resolver logaritmos discretos en

tiempo polinómico, haciendo que los métodos de clave pública actuales sean obsoletos ante una amenaza cuántica.

En respuesta a esta amenaza, el NIST ha recomendado la transición hacia algoritmos basados en retículos (lattice-based cryptography). Para firmas digitales, el estándar recomendado es ML-DSA (Module-Lattice-Based Digital Signature Algorithm), estandarizado en el FIPS 204 [24]. ML-DSA ofrece seguridad matemáticamente resistente a ataques cuánticos mientras mantiene una eficiencia computacional aceptable para la mayoría de aplicaciones, incluyendo dispositivos con recursos limitados.

Es importante destacar que, en términos de eficiencia de verificación, ML-DSA presenta un rendimiento competitivo frente a algoritmos tradicionales. Según benchmarks realizados en arquitecturas modernas x64 [1], la verificación de una firma ML-DSA es más rápida que la de una curva elíptica equivalente (ECDSA), lo cual es relevante para dispositivos IoT que actúan frecuentemente como verificadores en procesos de actualización OTA. Por otro lado, la generación de firmas presenta tiempos similares.

Algoritmo	**Firma (ms)**	**Verificación (ms)**
ECDSA (Level 5)	0.1754	0.3126
ML-DSA (Level 5)	0.0956	0.0479

Tabla 2.1: Comparativa de tiempos de firma y verificación en x64 (ms) [1]

Sin embargo, el punto crítico de ML-DSA radica en el tamaño de sus claves y firmas. A diferencia de ECDSA o RSA, ML-DSA requiere tamaños significativamente mayores. Por ejemplo, mientras una firma ECDSA P-256 ocupa 64 bytes, una firma ML-DSA-44 asciende a 2420 bytes ($\sim$2.4 KB). Este incremento impacta en el almacenamiento y el ancho de banda necesario para la transmisión de las firmas.

Si bien la transición no es urgente, dado que actualmente no existen computadores cuánticos capaces de ejecutar el algoritmo de Shor a escala práctica y todas las firmas verificadas hoy en día permanecen seguras, realizar el salto hacia algoritmos post-cuánticos es altamente recomendado para garantizar la seguridad a largo plazo.

La combinación de cifrado simétrico resistente a la computación cuántica (ASCON-80pq o AES-256-GCM) con firmas digitales basadas en retículos (ML-DSA) constituye un enfoque integral para garantizar la seguridad criptográfica a largo plazo, tanto ante amenazas actuales como futuras.

2.3 Actualizaciones de software en sistemas embebidos

La capacidad de actualizar el software de manera remota y segura (OTA - Over-The-Air) nos permiten, no solo corregir errores y vulnerabilidades de seguridad, sino también desplegar nuevas funcionalidades durante la vida útil del dispositivo.

Para comprender el diseño del sistema de actualizaciones propuesto, es necesario definir primero algunos conceptos clave del ecosistema Linux embebido.

2.3.1 Conceptos fundamentales

2.3.1.1. Yocto Project

El Proyecto Yocto es un proyecto de colaboración de código abierto que ayuda a los desarrolladores a crear sistemas personalizados basados en Linux para productos embebidos, independientemente de la arquitectura del hardware. Proporciona un conjunto flexible de herramientas y un espacio en el que los desarrolladores de sistemas embebidos de todo el mundo pueden compartir tecnologías, pilas de software, configuraciones y mejores prácticas que se pueden utilizar para crear imágenes de Linux personalizadas para dispositivos embebidos.

En la terminología de Yocto, se definen conceptos clave como *recipes* (recetas), que son archivos que describen cómo obtener, configurar, compilar e instalar un paquete de software, y *layers* (capas), que agrupan recetas relacionadas. Este sistema permite generar imágenes de sistema completas, incluyendo el bootloader, el kernel y el rootfs, de manera reproducible y controlada.

2.3.1.2. Bootloader

El *bootloader* o gestor de arranque es el primer programa que se ejecuta cuando se enciende el dispositivo. Su responsabilidad principal es inicializar el hardware esencial (CPU, memoria RAM, relojes) y cargar el sistema operativo (kernel de Linux) en la memoria para su ejecución.

En el contexto de las actualizaciones OTA, el bootloader juega un papel crítico: es el componente encargado de decidir qué partición del sistema operativo se debe arrancar. Esto permite, por ejemplo, arrancar desde una nueva versión del software tras una actualización exitosa o volver a la versión anterior si la nueva falla. U-Boot (Universal Boot Loader) es el estándar de facto en sistemas Linux embebidos debido a su flexibilidad y soporte para múltiples arquitecturas.

2.3.1.3. Rootfs (Sistema de archivos raíz)

El *rootfs* o sistema de archivos raíz es la estructura de directorios que contiene todos los archivos necesarios para que el sistema operativo funcione, incluyendo bibliotecas, binarios, archivos de configuración y las aplicaciones del usuario. En sistemas embebidos robustos, es común que el rootfs se monte en modo de solo lectura (*read-only*) durante la operación normal. Esto protege la integridad del sistema frente a cortes de energía repentinos o corrupción de datos, asegurando que el dispositivo siempre arranque en un estado conocido.

2.3.1.4. Estrategia de actualización A/B

La estrategia de actualización A/B, también conocida como de doble partición o *dual-copy*, es el mecanismo más robusto para garantizar actualizaciones seguras y atómicas. El almacenamiento del dispositivo se divide en dos conjuntos idénticos de particiones (Slot A y Slot B) para el sistema operativo (kernel y rootfs).

El funcionamiento es el siguiente:

- El dispositivo arranca y opera desde una de las particiones (por ejemplo, Slot A), que se considera la partición .ªctiva".
- Cuando se recibe una actualización, esta se instala en la partición inactiva (Slot B), sin interrumpir el funcionamiento del sistema.
- Una vez completada la instalación, se instruye al bootloader para que intente arrancar desde el Slot B en el próximo reinicio.
- Si el arranque del Slot B es exitoso, se marca como la nueva partición activa. Si falla (por ejemplo, el sistema se cuelga o no pasa los tests de arranque), el bootloader revierte automáticamente al Slot A, garantizando que el dispositivo siga operativo.

Este mecanismo proporciona redundancia y asegura que el dispositivo nunca quede inutilizado por una actualización fallida o corrupta, como se ilustra en la Figura 2.12.

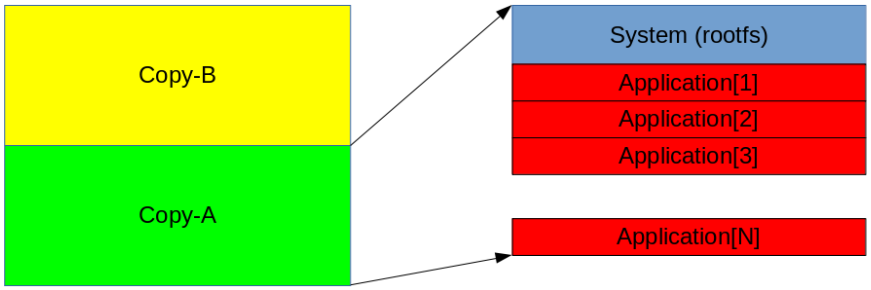

Figura 2.12: *Proceso de instalación de actualización A/B.*

2.3.2 SWUpdate: Gestor de actualizaciones

SWUpdate es un framework de actualización de software de código abierto diseñado específicamente para sistemas Linux embebidos. A diferencia de gestores de paquetes tradicionales como *apt* o *dnf*, que actualizan archivos individuales, SWUpdate está orientado a la actualización de imágenes completas del sistema o particiones, lo cual es ideal para garantizar la consistencia en dispositivos IoT. Cabe destacar que SWUpdate es una de las soluciones recomendadas por el Proyecto Yocto para realizar actualizaciones de sistema completas y seguras [25].

SWUpdate actúa como un agente que se ejecuta en el dispositivo y gestiona todo el proceso de actualización. Sus principales características y cualidades incluyen:

- **Soporte para actualizaciones A/B:** Se integra nativamente con el bootloader (como U-Boot) para gestionar el cambio de particiones y el mecanismo de *fallback* en caso de fallo.
- **Actualizaciones atómicas:** Garantiza que el sistema se actualice completamente o no se actualice en absoluto, evitando estados inconsistentes.
- **Streaming:** Permite instalar la actualización mientras se descarga, sin necesidad de almacenar el archivo completo en el almacenamiento local. Esto es crucial en dispositivos con memoria flash limitada.
- **Seguridad:** Soporta la verificación de imágenes firmadas digitalmente (utilizando RSA, CMS o claves simétricas) y el descifrado de imágenes, asegurando que solo software auténtico y autorizado pueda ser instalado.
- **Flexibilidad:** Es altamente configurable y soporta múltiples fuentes de actualización (USB, red, OTA) y formatos de imagen (raw, comprimidos con zstd/gzip, etc.).

En este proyecto, SWUpdate se utiliza como el componente central en el lado del dispositivo para orquestar la descarga, verificación e instalación de las nuevas versiones de firmware.

2.3.2.1. Arquitectura modular

Una de las características más destacadas de SWUpdate es su arquitectura altamente modular. El sistema dispone de gran cantidad de módulos o ”handlers”que permiten gestionar diferentes tipos de artefactos (imágenes de disco, archivos, scripts, bootloaders) y comunicarse con diversos backends de gestión (SurBit, Hawkbit, servidores web genéricos, etc.).

En este proyecto en específico, se ha hecho uso del módulo **WFX** [26]. Este módulo actúa como unión con el gestor de actualizaciones central c, encargándose de la comunicación con la plataforma de gestión en la nube donde WFX está ejecutado. A través del módulo WFX, el dispositivo puede consultar la disponibilidad de nuevas versiones, descargar los paquetes de actualización y reportar el estado final del proceso. El módulo WFX tiene soporte para dos tipos de actualizaciones: directo y por fases, donde el segundo requiere actividad del desarrollador. De todos modos, esto se detallará más en profundidad en la siguiente sección en conjunto con la herramienta WFX.

2.3.2.2. Formato de paquete .swu y descriptor

El paquete de actualización que maneja SWUpdate es un archivo con extensión `.swu`. Técnicamente, un archivo `.swu` es un contenedor en formato CPIO (*Copy In, Copy Out*) que agrupa todos los elementos necesarios para la actualización. Dentro de este contenedor se encuentran:

- **Imágenes de software:** Los binarios, sistemas de archivos o archivos individuales que se van a instalar (por ejemplo, imágenes *raw* o archivos de firmware).

- **Scripts:** Scripts de pre-instalación o post-instalación si son necesarios.
- **Descriptor (sw-description):** El archivo más importante, que describe el contenido del paquete y las reglas de instalación.

El archivo `sw-description` utiliza una sintaxis basada en *libconfig* y define metadatos como la versión, compatibilidad de hardware y la lista de archivos a instalar. A continuación se muestra un ejemplo de un descriptor básico:

```
software =
{
        version = "1.2";
        reboot = false;
        description = "Update vnull - OTA patch";
        hardware-compatibility: [ "1.0", "1.2", "1.3" ];

        ecs = {
                files: (
                        {
                                filename = "fichero";
                                path = "/ruta/fichero";
                                type = "rawfile";
                        }
                );
        }
}
```

En este ejemplo, se define una actualización versión «1.2» compatible con varias revisiones de hardware. La sección `ecs` (que podría ser cualquier nombre de agrupación) contiene una lista de archivos, en este caso un único archivo de tipo `rawfile` que se copiará a la ruta especificada.

2.3.2.3. Flujo de actualización (Pulling)

El proceso de actualización en este proyecto sigue un modelo de ”pulling”(sondeo), donde la iniciativa parte del dispositivo. El funcionamiento típico con el gestor de actualizaciones es el siguiente:

1. **Comprobación (Polling):** El servicio SWUpdate en el dispositivo consulta periódicamente al servidor central (a través del módulo WFX) si existe una nueva actualización disponible para su versión de hardware y software actual.
2. **Descarga:** Si el servidor responde afirmativamente con una actualización pendiente, el dispositivo comienza la descarga del archivo `.swu`.

3. **Instalación:** SWUpdate procesa el archivo `.swu`, lee el descriptor `sw-description` y procede a instalar los artefactos según las reglas definidas.

4. **Confirmación:** Una vez finalizada la instalación, el dispositivo reporta el éxito o fracaso de la operación al servidor.

2.3.3 Generación de paquetes: SWUGenerator

Para la creación de los paquetes de actualización `.swu`, existe una herramienta complementaria denominada **SWUGenerator** [27]. Esta utilidad, escrita en Python y desarrollada por los mismos creadores de SWUpdate (Stefano Babic), simplifica significativamente el proceso de empaquetado.

SWUGenerator toma como entrada los archivos que componen la actualización (imágenes de sistema, scripts, etc.) y un archivo de configuración simplificado, y se encarga de construir automáticamente el contenedor CPIO dado el descriptor y ficheros.

Una de las características más potentes de SWUGenerator es su capacidad para integrar la seguridad en el proceso de construcción. Permite, mediante parámetros de línea de comandos o configuración, firmar digitalmente el paquete utilizando los mismos algoritmos criptográficos soportados por SWUpdate (RSA, CMS, etc.). Asimismo, cabe destacar que los mecanismos de cifrado y firma son los mismos que los utilizados en SWUpdate, garantizando la compatibilidad total entre la generación y la instalación de los paquetes. Esto supone que, de añadir nuevos algoritmos en SWUpdate, SWUGenerator también necesitaría ser actualizado para soportarlos.

2.3.4 WFX: Workflow Executioner

WFX (*Workflow Executioner*) es una herramienta desarrollada por Siemens y escrita en el lenguaje de programación Go, diseñada para gestionar la ejecución de flujos de trabajo (*workflows*) en entornos distribuidos. En el contexto de este proyecto, WFX actúa como el gestor del sistema de actualizaciones OTA, orquestando el ciclo de vida de las actualizaciones desde que se definen hasta que se instalan en los dispositivos.

El funcionamiento de WFX se basa en máquinas de estados cliente-servidor. Un flujo de trabajo define una serie de estados por los que debe pasar una tarea (en este caso, una actualización) y las transiciones permitidas entre ellos. Estas transiciones pueden ser desencadenadas por el dispositivo (el cliente, a través de SWUpdate) o por el operador del sistema (el desarrollador o administrador).

Para facilitar esta interacción, WFX expone dos APIs diferenciadas:

- **API Norte (North API):** Destinada al desarrollador o a los sistemas de gestión. Permite crear trabajos, monitorizar su estado y ejecutar transiciones administrativas (como aprobar una actualización).

- **API Sur (South API):** Destinada a los dispositivos. Es utilizada por el agente SWUpdate (a través del módulo WFX) para consultar tareas pendientes, reportar progreso y actualizar el estado

de la instalación.

2.3.4.1. Tipos de Workflows

WFX integra flujos de trabajo específicos para la actualización de artefactos en dispositivos, conocidos como DAU (*Device Artifact Update*). El sistema soporta dos modalidades principales: *Direct* (Directo) y *Phased* (Por fases).

El **Workflow Directo** es un proceso automatizado diseñado para despliegues rápidos donde no se requiere intervención manual. Una vez creado el trabajo, el dispositivo procede a la descarga e instalación sin pausas intermedias, como se muestra en la Figura 2.13.

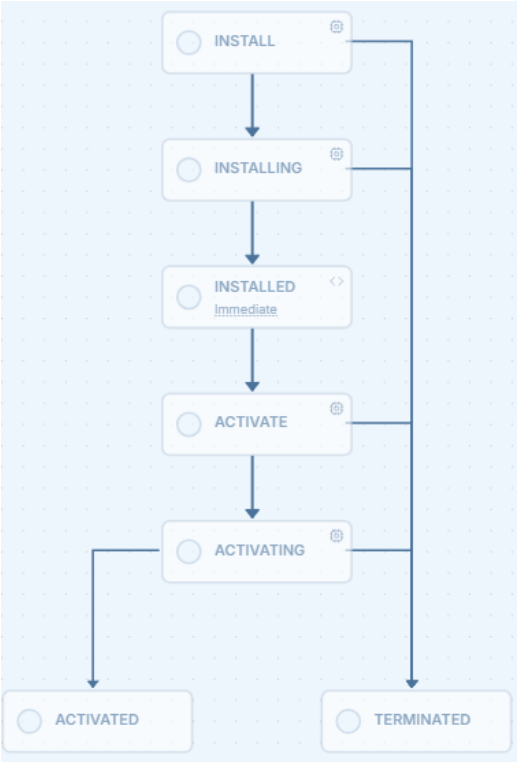

Figura 2.13: *Diagrama de estados del Workflow Directo.*

El **Workflow Por Fases** introduce puntos de control manual para una gestión más controlada del despliegue. Es fundamental notar los estados marcados con los símbolos <> en el diagrama de la Figura 2.14. Estos estados indican que la transición no es automática, sino que requiere una acción explícita por parte del desarrollador a través de la API Norte. Específicamente, estas intervenciones son necesarias para dar inicio a la actualización y para activarla una vez instalada, permitiendo un control granular sobre el despliegue.

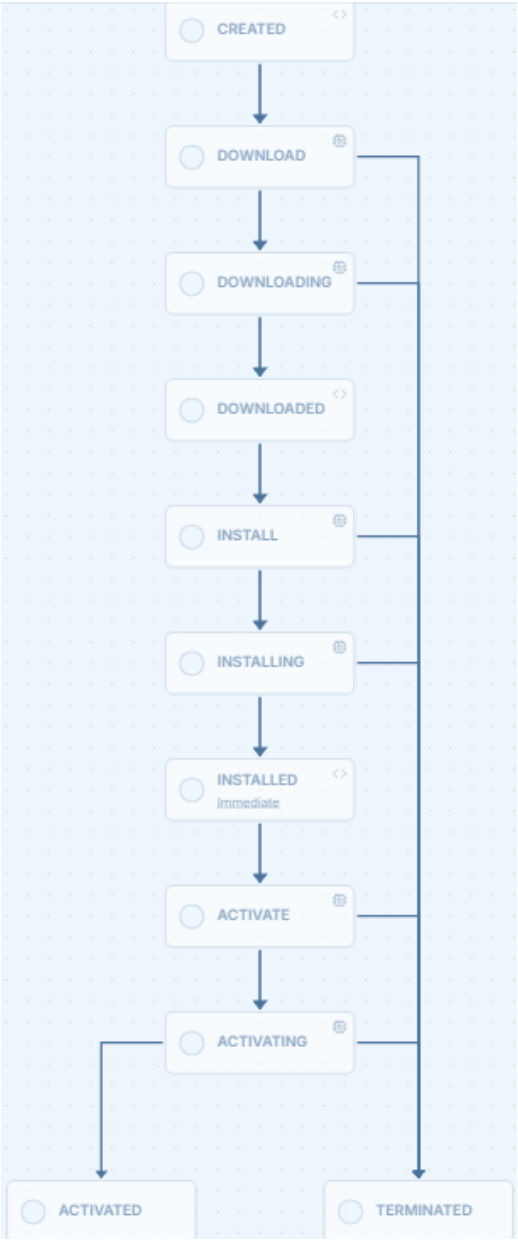

Figura 2.14: *Diagrama de estados del Workflow Por Fases.*

Además de los flujos de trabajo predefinidos, WFX permite la creación de **Workflows personalizados**. Estos flujos pueden adaptarse a necesidades específicas del proyecto, incorporando estados adicionales, transiciones únicas o integraciones con otros sistemas. La flexibilidad para definir workflows personalizados es una característica poderosa que permite a los desarrolladores diseñar procesos de actualización que se alineen perfectamente con los requisitos operativos y de seguridad de su entorno. Sin embargo, esto supone un esfuerzo adicional también en la implementación del lado del dispositivo, ya que el agente SWUpdate debe ser capaz de manejar los nuevos estados y transiciones definidos.

2.3.4.2. La Entidad Job

En la arquitectura de WFX, cuando se asigna una actualización a un dispositivo, se crea una entidad denominada **Job** (Trabajo). El agente SWUpdate en el dispositivo sondea constantemente la API Sur de WFX buscando si existe algún *Job* activo asignado a su *DeviceID*.

Un *Job* encapsula toda la información necesaria para realizar la actualización, tal y como se ilustra en la Figura 2.15. Sus componentes principales son:

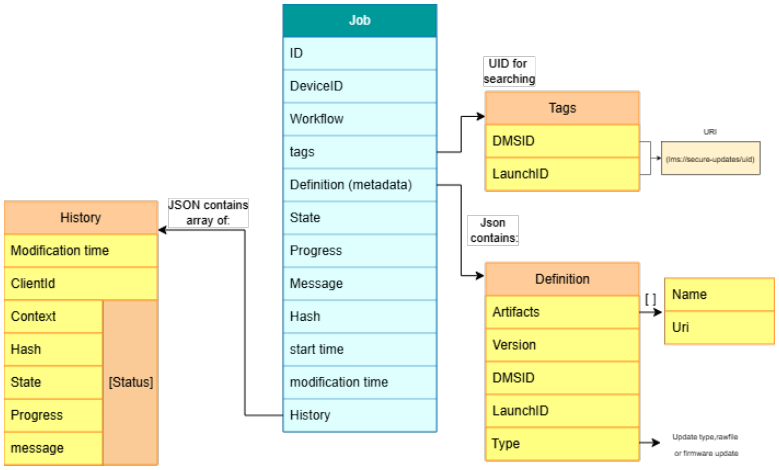

Figura 2.15: *Estructura de la entidad Job en WFX.*

- **ID:** Identificador único del trabajo.
- **DeviceID:** Identificador del dispositivo al que está asignado el trabajo.
- **Workflow:** Nombre del flujo de trabajo que rige este trabajo (ej. `wfx.workflow.dau.direct`).
- **Tags:** Etiquetas para categorización y búsquedas.
- **State:** Estado actual del trabajo dentro del flujo (ej. `CREATED`, `DOWNLOADING`, `INSTALLED`).
- **Progress:** Indicador numérico del progreso de la tarea.
- **Message:** Mensajes de log o errores reportados por el dispositivo.
- **Hash:** Valor hash para identificar y verificar la integridad del trabajo.
- **Fechas:** *Start time* (inicio) y *Modification time* (última actualización).
- **History:** Un registro histórico en formato JSON que almacena la evolución del trabajo, incluyendo cambios de estado, tiempos de modificación y mensajes.

Un componente crítico del *Job* es el campo **Definition** (metadatos). Este campo se almacena como un objeto JSON y contiene los detalles técnicos de la actualización que el dispositivo necesita procesar. En este proyecto, el campo *Definition* se ha utilizado para inyectar identificadores y configuraciones específicas:

- **Artifacts:** Lista de objetos a instalar. Cada artefacto incluye un **Name** y una **Uri** (ruta de descarga para el dispositivo).
- **Version:** Versión del paquete de software.
- **Type:** Tipo de actualización.
- **DMSID y LaunchID:** Etiquetas utilizadas como identificadores únicos para búsquedas y trazabilidad dentro del sistema de gestión.

2.3.5 LamassuIoT

Lamassu es una infraestructura de clave pública (PKI) diseñada específicamente para dispositivos IoT. Ofrece capacidades completas para la gestión de claves asimétricas, autoridades certificadoras (CA) y certificados digitales. Esta infraestructura se integra directamente con los dispositivos, permitiéndoles utilizar estos materiales criptográficos para procesos de autenticación robusta.

Un concepto central en Lamassu es el DMS (*Device Management System*). Un DMS actúa como una agrupación lógica de dispositivos, permitiendo administrar flotas de dispositivos de manera organizada. En el contexto de una arquitectura IoT, un DMS puede representar una zona geográfica, un tipo de dispositivo específico o un cliente particular.

En el alcance de este proyecto, Lamassu desempeña un papel fundamental como proveedor de identidad y seguridad:

- **Gestión de Dispositivos:** Provee las listas de dispositivos autorizados y sus agrupaciones (DMS).
- **Infraestructura de Confianza:** Suministra los mecanismos de firma digital y emisión de certificados necesarios para validar la autenticidad del firmware y de los propios dispositivos.

2.3.6 Extensión del KMS para Criptografía Simétrica

Es importante destacar que, en su estado actual, Lamassu no disponía de un sistema de gestión de claves (KMS) para criptografía simétrica. Como parte de los objetivos de este trabajo, se ha desarrollado esta capacidad, extendiendo el KMS de Lamassu para soportar:

- **Gestión de claves simétricas:** Generación, importación, almacenamiento y rotación de claves para algoritmos de cifrado simétrico (AES en sus variantes y ASCON).
- **Operaciones de cifrado/descifrado:** Ejecución de operaciones criptográficas directamente en el KMS, manteniendo las claves protegidas sin exponerlas.
- **Generación de MAC (Message Authentication Code):** Soporte para algoritmos de autenticación de mensajes, incluyendo AES-CMAC y ASCON-MAC, permitiendo verificar la integridad y autenticidad de los datos sin cifrarlos.

Esta extensión permite que el servicio de Updates utilice el KMS de Lamassu para todas las operaciones criptográficas necesarias en el proceso de creación de paquetes de actualización, garantizando una gestión centralizada y segura de las claves criptográficas.

2.4 Infraestructura Planteada

La arquitectura propuesta para la plataforma de actualizaciones seguras se ha diseñado con el objetivo de desacoplar la complejidad de la gestión de actualizaciones de la lógica de negocio, proporcionando una interfaz unificada y segura para los desarrolladores.

Como se ha mencionado anteriormente, **WFX** actúa como el motor de distribución de las actualizaciones. Es el componente encargado de mantener el estado de los despliegues y servir las actualizaciones a los dispositivos. En el lado del dispositivo, la herramienta **SWUpdate** es la responsable de realizar el *polling* periódico hacia WFX para comprobar si existen nuevas tareas asignadas, y en caso afirmativo, descargar e instalar la actualización siguiendo el flujo definido.

Para orquestar todo este proceso y facilitar su gestión, se introduce un nuevo servicio denominado **Updates**. Este módulo actúa como una capa de abstracción y control sobre WFX. Sus responsabilidades principales incluyen:

- **Construcción de Paquetes:** Utilizar la herramienta **SWUGenerator** para crear los paquetes de actualización a partir de los ficheros proporcionados por el desarrollador, generando el formato SWU requerido por SWUpdate.
- **Gestión de Versiones:** Administrar el catálogo de versiones de software disponibles.
- **Gestión de Despliegues:** Coordinar la creación y supervisión de las campañas de actualización.
- **Interacción con WFX:** Comunicarse con la API de WFX para crear los trabajos (*Jobs*) y monitorizar su progreso, ocultando esta complejidad al usuario final.
- **Integración con Lamassu:** Utilizar los servicios de **KMS** (Key Management System) para las operaciones criptográficas (cifrado y firma de paquetes) y el servicio **DMS** (Device Management System) para obtener la información de los dispositivos y sus agrupaciones.

De esta manera, el desarrollador puede desplegar actualizaciones utilizando este servicio sin necesidad de interactuar directamente con los detalles de bajo nivel de WFX o la gestión de claves.

Adicionalmente, se ha desarrollado una **Interfaz Web** que proporciona una experiencia de usuario visual para la gestión del sistema. Esta interfaz se comunica con la API REST expuesta por el servicio de Updates, permitiendo a los administradores subir nuevos firmwares, definir estrategias de actualización y visualizar el estado de la flota de dispositivos en tiempo real.

La arquitectura global de la solución y la interacción entre estos componentes se ilustra en la Figura 2.16.

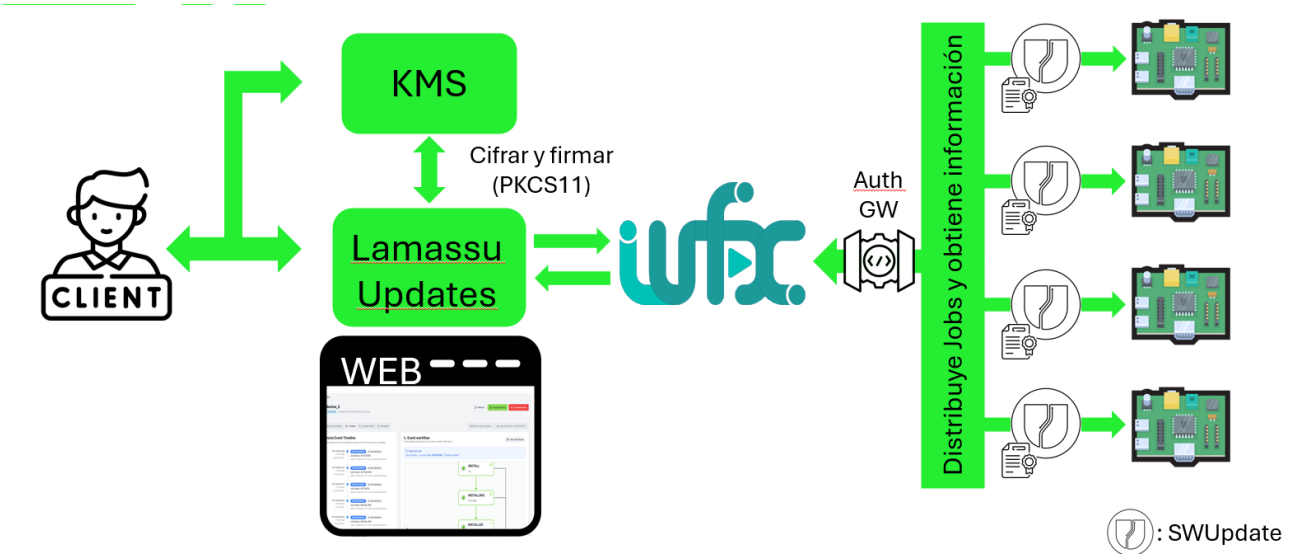

Figura 2.16: *Arquitectura de la plataforma de actualizaciones propuesta*

2.5 Integración de Criptografía Ligera y Post-Cuántica

Una de las contribuciones principales de este trabajo es la integración de algoritmos criptográficos ligeros (LWC) y resistentes a ataques cuánticos (PQ) en la cadena de actualización de dispositivos IoT. Esta integración ha requerido modificaciones tanto en SWUpdate como en SWUGenerator, presentando desafíos técnicos considerables.

2.5.1 Integración de Algoritmos de Cifrado Simétrico

La incorporación de nuevas alternativas de criptografía simétrica, en particular ASCON, si bien swupdate es muy personalizable, este cambio ha supuesto una dificultad considerable. Originalmente, SWUpdate únicamente admitía cifrado simétrico mediante AES-CBC, sin ningún mecanismo de selección de algoritmo. Esta restricción se debía a que, al existir una sola opción, no se requería un selector explícito en el diseño original.

Para superar esta limitación, se ha modificado el código fuente de SWUpdate para incorporar las siguientes capacidades:

- **Selector de algoritmo:** Posibilidad de definir explícitamente qué algoritmo de cifrado se utilizará.
- **Soporte para variantes de AES:** Integración de las variantes CBC, CTR y GCM de AES para las longitudes de clave de 128, 192 y 256 bits.
- **Soporte para ASCON:** Integración de la implementación en C de ASCON en su variante más eficiente.

Por defecto, el sistema asume que las claves criptográficas se encuentran almacenadas en `/etc/swupdate/keys/a` siguiendo una convención de nomenclatura que incluye el algoritmo con sub variante, el IV y la clave.

2.5.1.1. Instalación Nativa

Para mantener la flexibilidad y no cerrar puertas a diferentes escenarios de uso, se ha conservado la opción de instalación nativa. Esto puede ocurrir si se requiere de una instalación mediante USB practica muy común en el ambito de dispositivos linux embebidos. En este modo, el algoritmo y la clave se especifican directamente por línea de comandos:

```
swupdate -i test_aes_192_ctr.swu -K ./aes192-ctr.txt
```

2.5.1.2. Integración con WFX

En el caso de despliegues orquestados mediante WFX, la información del algoritmo de cifrado se transmite a través de los metadatos del Job. Al recibir una tarea pendiente, el agente SWUpdate extrae el algoritmo especificado en el campo *definition*, que WFX proporciona para usos personalizados.

El formato de metadatos utilizado en los Jobs de WFX es el siguiente:

Listing 2.1: Formato de metadatos del Job en WFX

```
{
  "id": "update-001",
  "definition": {
    "artifacts": [
      {
        "uri": "https://cdn.example.com/firmware.swu.enc",
        "ivt": "0123456789abcdef0123456789abcdef",
        "algorithm": "aes256-cbc"
      }
    ]
  }
}
```

Los campos clave son:

- **uri:** Ubicación desde donde el dispositivo descargará el paquete de actualización.
- **ivt:** Vector de inicialización utilizado en el cifrado.
- **algorithm:** Algoritmo de cifrado empleado (ej. `aes256-cbc`, `ascon-128a`).

2.5.2 Integración de Firma Digital Post-Cuántica

La integración de mecanismos de firma digital resistentes a ataques cuánticos resultó relativamente más sencilla que la del cifrado simétrico. El proceso requirió dos modificaciones principales:

1. **Habilitación en menuconfig:** SWUpdate utiliza un sistema de configuración basado en Kconfig (similar al kernel de Linux) que permite habilitar o deshabilitar características antes de compilar (*buildear*) el proyecto. Se habilitaron los métodos de firma en esta configuración, en especifico verificación de firma con formato CMS.
2. **Actualización de OpenSSL:** Para el soporte de algoritmos post-cuánticos, se requiere OpenSSL versión 3.5 o superior. Dado que los repositorios de paquetes (APT) no disponen aún de esta versión, fue necesario descargarla del repositorio oficial de OpenSSL y realizar una instalación manual.

2.5.3 Modificaciones en SWUGenerator

La integración de los nuevos algoritmos criptográficos también requirió modificaciones sustanciales en SWUGenerator, la herramienta responsable de crear los paquetes SWU.

2.5.3.1. Cifrado con ASCON

Inicialmente, se evaluó el uso de la librería de Python de ASCON [28], disponible en PyPI. Sin embargo, las pruebas de rendimiento revelaron que esta implementación tardaba cientos de veces más que una implementación nativa, requiriendo varios segundos para procesar apenas unos pocos megabytes. Ante esta limitación de rendimiento inaceptables para paquetes de firmware que pueden ser de decenas o cientos de megabytes, se optó por integrar directamente el código C de ASCON, obteniendo una mejora drástica en el tiempo de procesamiento.

2.5.3.2. Métodos de Firma

Se han integrado dos modalidades de firma digital en SWUGenerator:

- **Firma directa:** Acceso directo a la clave privada almacenada en el sistema de archivos. Este método es más simple pero requiere que la clave privada esté accesible en texto plano.
- **Firma CMS con PKCS#11:** Utilización del estándar PKCS#11 (*Public-Key Cryptography Standards #11*), que define una interfaz de programación independiente de la tecnología para dispositivos criptográficos como módulos de seguridad hardware (HSM). Este método permite que las claves privadas permanezcan protegidas dentro de hardware especializado, sin exponerlas nunca al sistema operativo.

- **Firma custom con certificado CMS provenientes de PKI**: Para este caso, se ha utilizado la API de Lamassu especificando el identificador de la clave privada y el certificado asociado a esa clave privada para crear el CMS que contenga la firma y el certificado. Esto proporciona una capa adicional de seguridad y autenticidad, ya que el certificado puede ser verificado por el dispositivo durante la instalación del paquete, aprovechando la infraestructura de gestión de claves de Lamassu.

2.5.3.3. Comando Completo de Generación

El comando completo para crear un paquete SWU con cifrado y firma integrados es el siguiente:

```
swugenerator \
  -k "CMS_PKCS11,pkcs11:object=key-name,cert.pem" \
  create \
  -K "key,iv,ascon-128a" \
  -t \
  -s sw-description \
  -o output.swu \
  -a artifacts/
```

Donde:

- `-k`: Especifica el método de firma (CMS con PKCS#11), el objeto de la clave en el dispositivo criptográfico y el certificado asociado.
- `-K`: Define la clave de cifrado, el vector de inicialización y el algoritmo a utilizar (en este caso, ASCON-128a).
- `-t`: Cifra el descriptor también.
- `-s`: Especifica el archivo descriptor del software.
- `-o`: Define el nombre del archivo de salida.
- `-a`: Indica el directorio que contiene los artefactos a empaquetar.

Con esto se habria realizado uno de los hitos de este TFM, que es la integración de criptografía ligera y post-cuántica en la cadena de actualización.

2.5.3.4. Proceso de Creación y Verificación de Actualizaciones Seguras

En el caso de integración con Lamassu, el proceso completo de creación y verificación de actualizaciones seguras sigue estos pasos:

1. **Creación en el backend:** El usuario sube los archivos de actualización a través de la interfaz web, selecciona la clave privada y su certificado asociado desde Lamassu, y elige la clave simétrica para cifrar la actualización. SWUGenerator crea el paquete cifrando tanto la actualización como el descriptor, y firma el conjunto usando el método custom con CMS, incorporando el certificado en la estructura CMS.
2. **Verificación en el dispositivo:** El dispositivo, con SWUpdate configurado previamente para validar firmas y cifrado, verifica la firma del paquete examinando el certificado incluido en el CMS contra la CA raíz instalada. Una vez validada la firma, descifra el contenido utilizando la clave privada correspondiente al algoritmo de cifrado especificado en el paquete.

Este flujo garantiza la integridad, autenticidad y confidencialidad de las actualizaciones, aprovechando la infraestructura PKI de Lamassu para la gestión segura de claves y certificados.

2.6 Modelo de Datos

Para entender el modulo de updates y su integración con Lamassu y WFX, es fundamental definir las entidades principales que componen el modelo de datos del sistema. Estas entidades representan los conceptos clave y las estructuras de información que se manejan en la plataforma de actualizaciones.

2.6.1 Entidades de Lamassu IoT

Desde la infraestructura Lamassu se obtienen dos entidades fundamentales que proporcionan la base organizativa y de identidad del sistema:

- **Device (Dispositivo):** Representa cada dispositivo IoT registrado en el sistema. Cada dispositivo contiene información de identificación, credenciales y metadatos asociados a su configuración de seguridad.
- **DMS (Device Management System):** Agrupa lógicamente conjuntos de dispositivos. Un DMS puede representar una flota de dispositivos de un cliente específico, una zona geográfica determinada, o cualquier otra agrupación organizativa. Esta entidad es clave para aplicar políticas de actualización a grupos completos de dispositivos.

2.6.2 Entidades de WFX

El sistema de orquestación de flujos de trabajo WFX aporta las siguientes entidades:

- **Workflow:** Define el flujo de estados por los que pasa una actualización. Como se ha descrito anteriormente, existen workflows predefinidos como *Direct* y *Phased*, y es posible crear workflows personalizados adaptados a necesidades específicas.

- **Job:** Representa una instancia de actualización asignada a un dispositivo concreto. Encapsula toda la información necesaria para llevar a cabo la actualización, incluyendo el estado actual, progreso, historial de cambios, y los metadatos técnicos de la actualización en su campo *Definition*.

2.6.3 Entidades del Módulo Updates

El módulo de Updates introduce dos entidades primordiales que gestionan el ciclo de vida de las actualizaciones:

2.6.3.1. Update-Pack (Paquete de Actualización)

El **Update-Pack** es la entidad que representa una actualización disponible en el sistema. Contiene toda la información necesaria para construir, cifrar y firmar el paquete de actualización. Sus campos principales son:

- **Name:** Nombre identificativo del paquete de actualización.
- **Version:** Versión del software o firmware contenido en el paquete.
- **Type:** Tipo de actualización, que puede ser *firmware* (actualización completa del sistema embebido) o *file* (fichero específico o componente de software).
- **Alg_Enc:** Algoritmo de cifrado utilizado para proteger el contenido del paquete (ej. AES-256, ASCON).
- **Alg_Sign:** Algoritmo de firma digital utilizado para garantizar la autenticidad e integridad del paquete (ej. RSA, ECDSA).
- **IV (Initialization Vector):** Vector de inicialización utilizado en el proceso de cifrado simétrico.
- **Descriptor_Encrypted:** Indicador booleano que especifica si el descriptor del paquete está cifrado .

Esta entidad permite mantener un catálogo versionado de actualizaciones, facilitando el control de versiones y la trazabilidad de cada paquete desplegado.

2.6.3.2. Launch (Lanzamiento)

El **Launch** representa una campaña de despliegue de una versión específica de actualización. A diferencia del Update-Pack que define *qué* se actualiza, el Launch define *cómo* y *cuándo* se despliega. Sus campos principales son:

- **ID:** Identificador único del lanzamiento.

- **ExecDate:** Fecha y hora de ejecución programada del lanzamiento.
- **Workflow_Type:** Tipo de workflow utilizado para este lanzamiento (ej. *Direct*, *Phased*, o un workflow personalizado).
- **Devices_Completed:** Número de dispositivos que han completado exitosamente la actualización.
- **Devices_Not_Started:** Número de dispositivos que aún no han iniciado el proceso de actualización.
- **Active_Devices:** Número de dispositivos que están actualmente ejecutando la actualización.
- **Rollout:** Estrategia de despliegue, que puede ser por porcentaje (%) o cantidad fija (*fixed*).
- **Rollout_Value:** Valor asociado a la estrategia de rollout (porcentaje o número de dispositivos).
- **Rollout_Mode:** Modo de progresión entre batches, que puede ser *auto* (automático) o *manual*.
- **Test_Device_ID:** Identificador del dispositivo utilizado para pruebas piloto antes del despliegue.
- **Launched_Version:** Referencia a la versión del Update-Pack que se está desplegando.

El Launch permite implementar estrategias de despliegue gradual (*phased rollout*), minimizando riesgos al actualizar primero un subconjunto de dispositivos antes de extender la actualización a toda la flota.

Las actualizaciones se despliegan organizadas en batches (lotes) de dispositivos según la estrategia definida en el campo *Rollout*. El sistema ofrece dos modos de progresión entre batches:

- **Modo Automático:** Una vez que todos los dispositivos de un batch completan exitosamente la actualización, el sistema procede automáticamente a lanzar el siguiente batch sin intervención manual. Este modo es ideal para despliegues donde se confía en la estabilidad de la actualización y se desea una distribución rápida.
- **Modo Manual:** Tras la finalización de cada batch, el sistema espera confirmación explícita del desarrollador o administrador antes de proceder con el siguiente lote. Este enfoque proporciona un mayor control y permite verificar el comportamiento de la actualización en producción antes de expandir el despliegue, siendo especialmente útil para actualizaciones críticas o de alto riesgo.

2.6.4 Relaciones entre Entidades

Las relaciones entre las entidades del sistema se establecen de la siguiente manera:

- Un **DMS** puede tener asociados múltiples **Update-Packs**, permitiendo gestionar diferentes versiones de software para una misma agrupación de dispositivos.
- Un **DMS** puede tener múltiples **Launches**, posibilitando campañas de actualización sucesivas o paralelas para diferentes componentes.
- Un **Launch** está asociado a un único **Update-Pack**, estableciendo una relación unívoca entre cada campaña de despliegue y la versión de software que distribuye.
- Cada **Job** creado en WFX está vinculado a un único dispositivo que forme parte del lanzamiento, por lo que en un lanzamiento finalizado habra tantos jobs como dispositivos planeados a actualizar.

La Figura 2.17 ilustra el diagrama completo del modelo de datos, mostrando cómo se relacionan las entidades del módulo de Updates con las entidades de Lamassu y WFX.

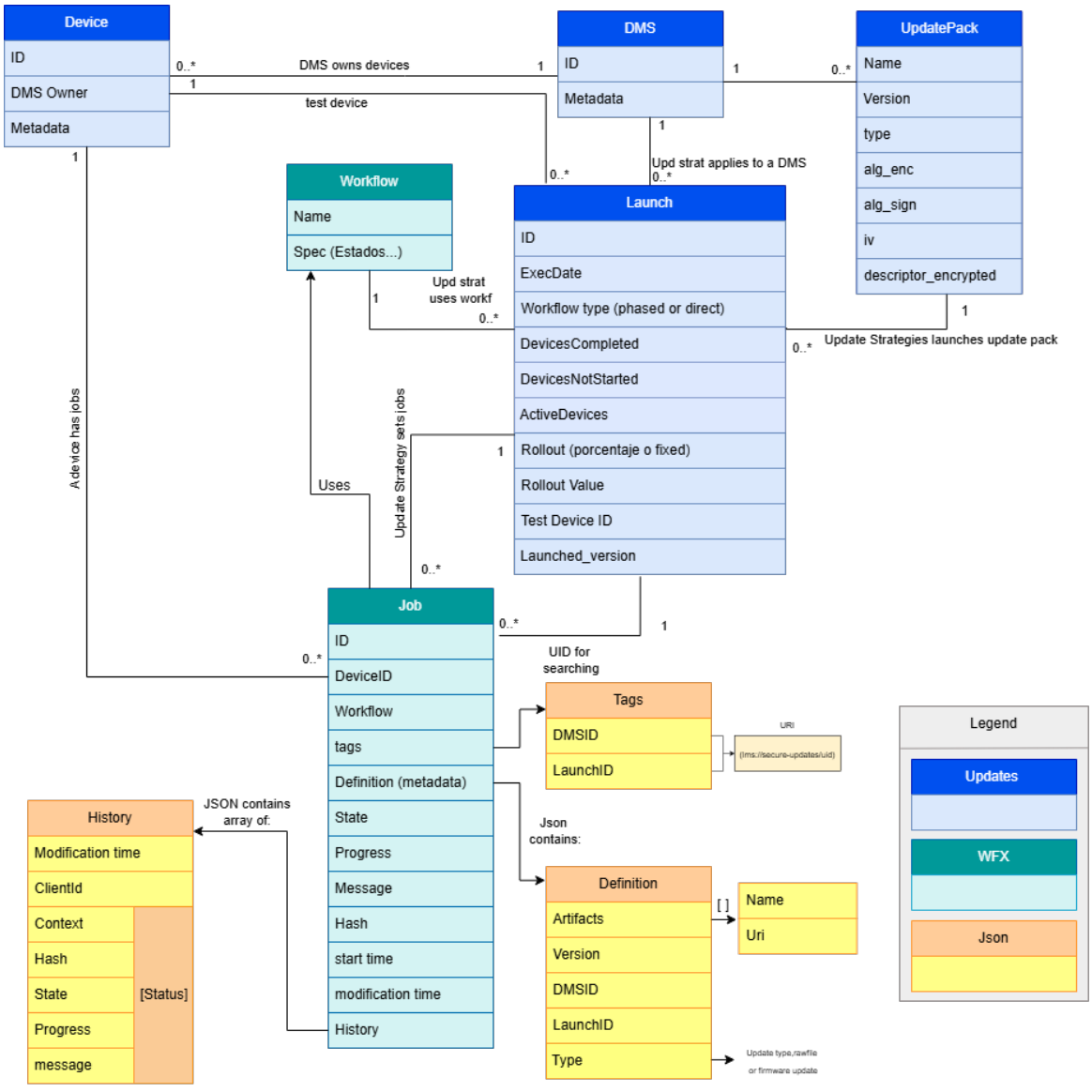

Figura 2.17: *Diagrama del modelo de datos de la plataforma*

2.7 Estrategia de Pruebas y Validación

Para garantizar la fiabilidad y correctitud de los dos componentes principales desarrollados, SymKMS (servicio de gestión de claves simétricas) y Updates (servicio de gestión de actualizaciones), se ha diseñado una estrategia de pruebas exhaustiva que abarca tests unitarios, de integración y validación criptográfica contra vectores oficiales.

2.7.1 Infraestructura de Tests

La ejecución de los tests de integración requiere servicios externos reales (PostgreSQL, WFX). Para garantizar la reproducibilidad y el aislamiento de las pruebas, se ha implementado una infraestructura basada en contenedores Docker mediante la librería `dockertest`. Esta infraestructura levanta automáticamente contenedores limpios de PostgreSQL 15 y WFX (Siemens Workflow Executor) antes de cada suite de tests, asigna puertos dinámicos para evitar conflictos y elimina los contenedores tras la finalización de las pruebas. De este modo, cada ejecución parte de un estado conocido e idéntico, independientemente del entorno.

2.7.2 Tests del Servicio SymKMS

El servicio SymKMS cuenta con más de 150 tests distribuidos en cuatro categorías principales:

2.7.2.1. Tests Unitarios del Servicio Core

Los tests unitarios del servicio core validan las operaciones criptográficas fundamentales:

- **Gestión de claves (15 tests):** Validación del ciclo de vida completo de las claves (creación, lectura, listado, eliminación), incluyendo paginación, ordenación y aislamiento multi-usuario para garantizar que las claves de un usuario no son accesibles por otro.
- **Creación de claves (9 tests):** Cobertura de todas las variantes soportadas: AES (128/192/256 bits en modos CBC, CTR y GCM), ASCON (128, 128a, 80pq), normalización de nombres de algoritmo y validación de tamaños de clave.
- **Cifrado y descifrado (21 tests):** Pruebas de *round-trip* (cifrar y descifrar recuperando el mensaje original) para todos los algoritmos. Se incluyen tests de detección de corrupción del *ciphertext* (especialmente relevantes para GCM y ASCON al ser modos AEAD), validación de *padding* en CBC, manejo de vectores de inicialización incorrectos y aislamiento por usuario.
- **Operaciones MAC (40 tests):** Validación de los tres algoritmos de MAC soportados: AES-CMAC con todos los tamaños de clave, HMAC-SHA3 (256, 384 y 512 bits) y ASCON-MAC. Se evalúa el determinismo (mismo input produce mismo MAC), comportamiento con datos de gran tamaño (hasta 1 MB), y casos límite como datos vacíos, de 1 byte, de 15 bytes (borde de

bloque AES) y múltiplos exactos del tamaño de bloque. También se prueban patrones especiales (*all-zeros*, *all-ones*).

- **Verificación de MAC (12 tests):** Validación de verificación correcta e incorrecta, detección de modificación de datos y detección de *tags* truncados o alterados.

2.7.2.2. Validación con Vectores de Test Oficiales

La correctitud criptográfica se ha validado contra fuentes de referencia:

- **AES-CMAC según RFC 4493 (4 tests oficiales):** Se han implementado los cuatro vectores de test definidos en la RFC 4493, cubriendo mensajes de 0, 16, 40 y 64 bytes. Durante este proceso se detectaron dos errores tipográficos en la propia RFC (discrepancias en los últimos bytes del vector 1 y tag incorrecto en el vector 4). Para confirmar la correctitud de la implementación, se realizó una validación cruzada con la librería `cryptography` de Python, obteniendo coincidencia del 100 %.
- **AES-CMAC condiciones de borde (10 tests):** Tests con longitudes de datos diseñadas para ejercitar todos los caminos del algoritmo de *padding*: 0, 1, 15, 16, 17, 32, 33, 48, 1024 y 65536 bytes.
- **AES-CMAC propiedades de seguridad (21 sub-tests):** Verificación de detección de *bit-flip* (primer y último byte), manipulación de *tag* y resistencia general a ataques de modificación.

2.7.2.3. Tests de API REST

Finalmente, se incluyen tests de los controladores HTTP que validan el correcto funcionamiento de todos los *endpoints* REST expuestos por el servicio, así como su input.

2.7.3 Tests del Servicio Updates

El servicio de Updates cuenta con más de 70 tests distribuidos en las siguientes categorías:

2.7.3.1. Tests Unitarios del Servicio

Los tests unitarios (10 tests) validan la lógica de negocio del servicio, incluyendo la creación de modelos *UpdatePack* con y sin cifrado y firma, la estructura correcta de los archivos SWU generados, y las combinaciones de algoritmos de cifrado (AES-256-CBC, AES-256-GCM, ASCON-128, ASCON-128a, ASCON-80pq) y firma (RSA-PSS, ECDSA, ML-DSA post-cuántico).

2.7.3.2. Tests de Integración con PostgreSQL

Los tests de integración con base de datos real (más de 25 tests) verifican las operaciones CRUD completas para las entidades *UpdatePack* y *LaunchTrack*, consultas avanzadas (por DMS ID, por nombre, con paginación), almacenamiento y recuperación de metadatos de cifrado y firma, concurrencia (10 inserciones paralelas) y todas las estrategias de despliegue (*Phased+Percentage*, *Direct+Fixed*, *Auto-deploy*).

2.7.3.3. Tests de Integración con WFX

Los tests de integración con WFX real (más de 15 tests) validan la carga de *workflows* en WFX, la creación y gestión de *jobs* a través de su API, las estrategias de lanzamiento con la base de datos real, la paginación del historial de actualizaciones y el versionado de *UpdatePacks*.

3. Resultados

Como resultado de este trabajo se ha desarrollado una interfaz web completa para gestionar todo el proceso de actualizaciones de dispositivos IoT, así como una demostración práctica donde validar la arquitectura propuesta. Esta sección presenta el caso de uso implementado y describe en detalle las funcionalidades de la interfaz web desarrollada.

3.1 Caso de Uso: Sistema de Control de Tanques

Para demostrar las capacidades de la plataforma, se ha implementado un caso de uso basado en un sistema de control de tanques de agua. El escenario planteado simula un problema real: un error en el software de control provoca que los tanques se desborden, representando un fallo crítico que debe corregirse mediante una actualización remota.

En este ejemplo, el software defectuoso está implementado como una aplicación HTML simple que simula el comportamiento de los sensores y actuadores del tanque. El objetivo es actualizar todos los dispositivos afectados a una versión corregida que solucione el problema de desbordamiento.

La demostración utiliza dos dispositivos virtuales, cada uno ejecutando una instancia del sistema de control.

La Figura 3.1 muestra la interfaz del sistema con el error de desbordamiento, mientras que la Figura 3.2 presenta la versión corregida tras la actualización.

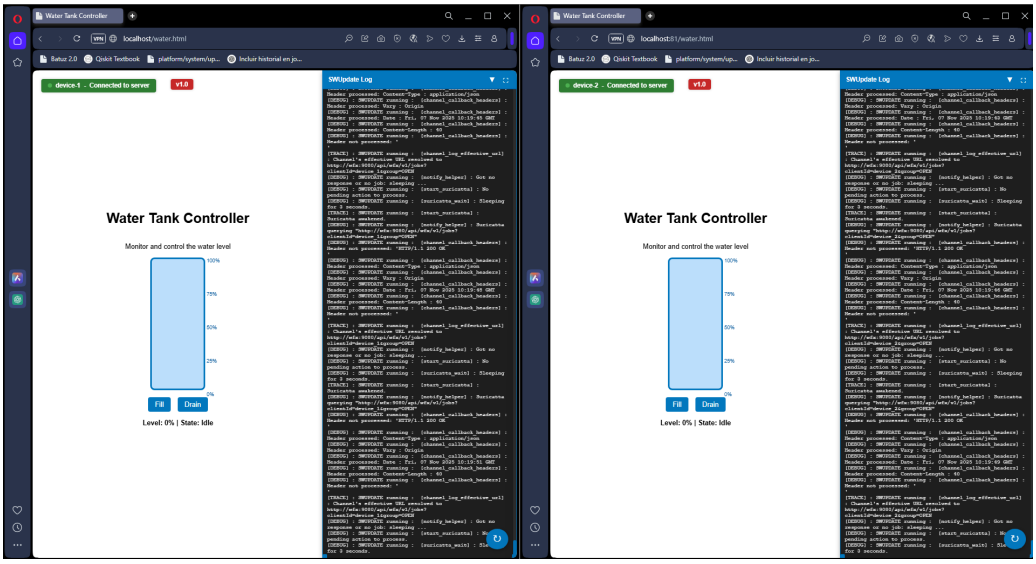

Figura 3.1: *Sistema de control de tanques con error de desbordamiento*

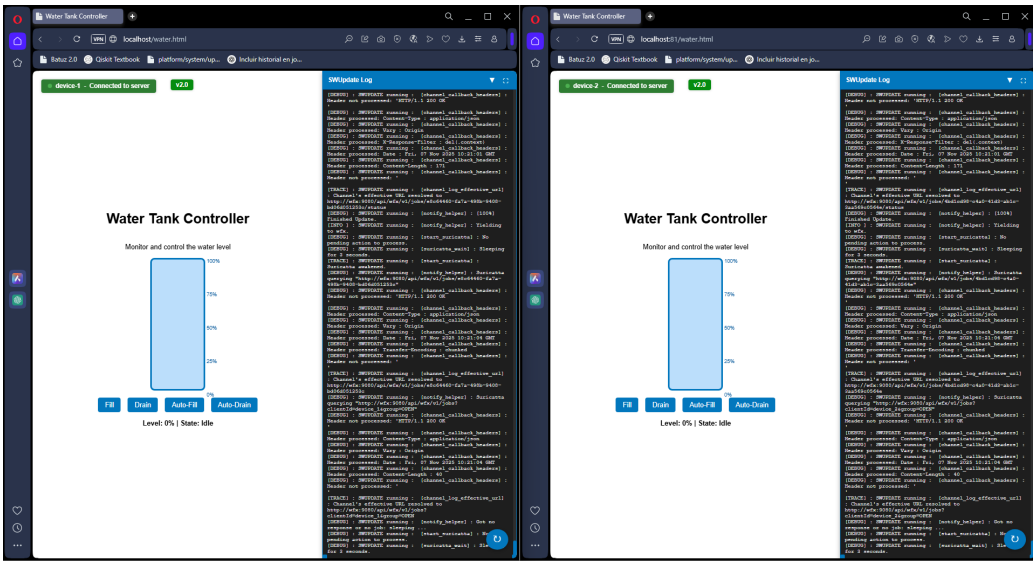

Figura 3.2: *Sistema de control de tanques versión corregida*

3.2 Interfaz Web de Gestión

La interfaz web desarrollada proporciona una experiencia de usuario completa para la gestión de claves criptográficas, paquetes de actualización y campañas de despliegue. A continuación se describen las principales funcionalidades implementadas.

3.2.1 Gestión de Claves Criptográficas (KMS)

El módulo KMS (*Key Management System*) constituye el punto de entrada para la gestión de las claves criptográficas utilizadas en el cifrado y firma de las actualizaciones.

3.2.1.1. Vista General del KMS

La página principal del KMS, mostrada en la Figura 3.3, permite visualizar todas las claves disponibles en el sistema. Desde esta interfaz, el usuario puede realizar las siguientes operaciones:

- **Crear nuevas claves:** Generación de claves simétricas (AES, ASCON).
- **Importar claves existentes:** Carga de claves previamente generadas desde archivos locales.
- **Visualizar metadatos:** Inspección de las propiedades de cada clave (algoritmo, longitud, fecha de creación, etc.).

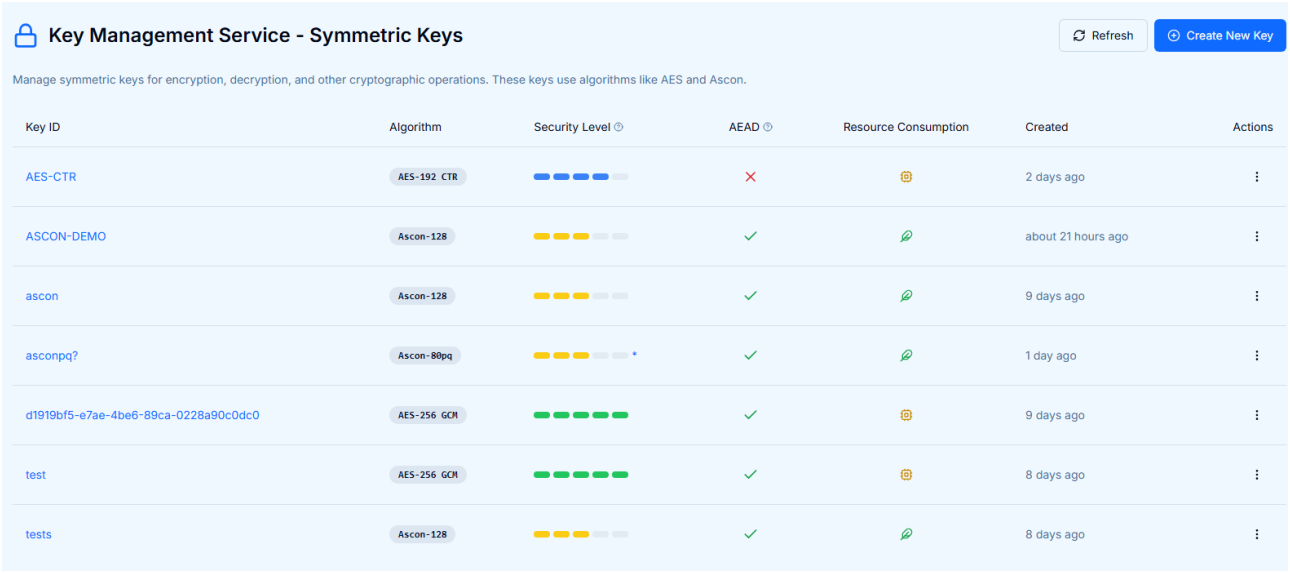

Figura 3.3: *Vista general del sistema de gestión de claves (KMS)*

3.2.1.2. Detalles y Operaciones de una Clave

Al seleccionar una clave específica, se accede a una vista detallada (ver Figura 3.4) que proporciona información completa sobre la clave y permite realizar operaciones criptográficas:

- **Cifrado:** Cifrado de datos utilizando la clave seleccionada.
- **Descifrado:** Descifrado de datos previamente cifrados con esta clave.
- **MAC (Message Authentication Code):** Generación de códigos de autenticación de mensajes para verificar la integridad de los datos.

Estas operaciones permiten validar el correcto funcionamiento de las claves antes de utilizarlas en el proceso de actualización.

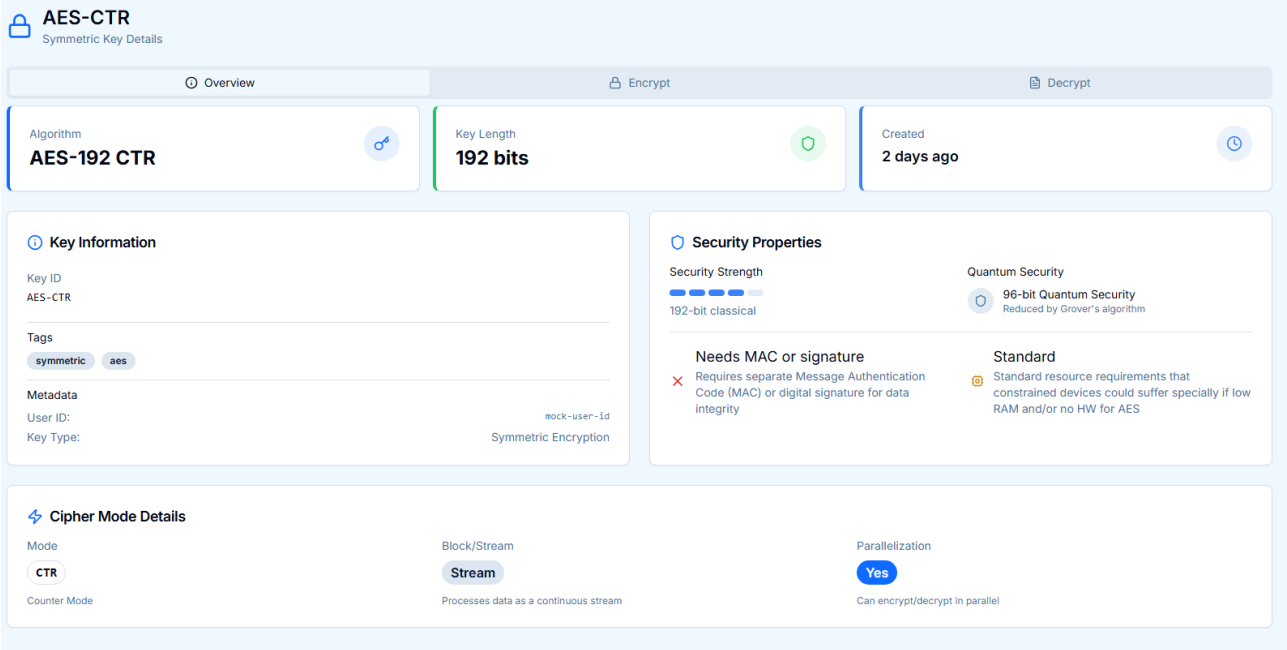

Figura 3.4: *Vista detallada de una clave con operaciones disponibles*

3.2.2 Gestión de Actualizaciones

3.2.2.1. Página Principal de Actualizaciones

La página principal del módulo de actualizaciones (ver Figura 3.5) presenta una vista consolidada de todos los lanzamientos agrupados por paquete de actualización (Update Pack). Esta interfaz permite:

- **Crear nuevas actualizaciones:** Iniciar el proceso de creación de un nuevo paquete de actualización.
- **Crear nuevas versiones:** Añadir versiones adicionales a un paquete existente.
- **Lanzar actualizaciones:** Iniciar una campaña de despliegue para una versión específica.
- **Visualizar el estado:** Monitorizar el progreso de las campañas activas y el histórico de despliegues.

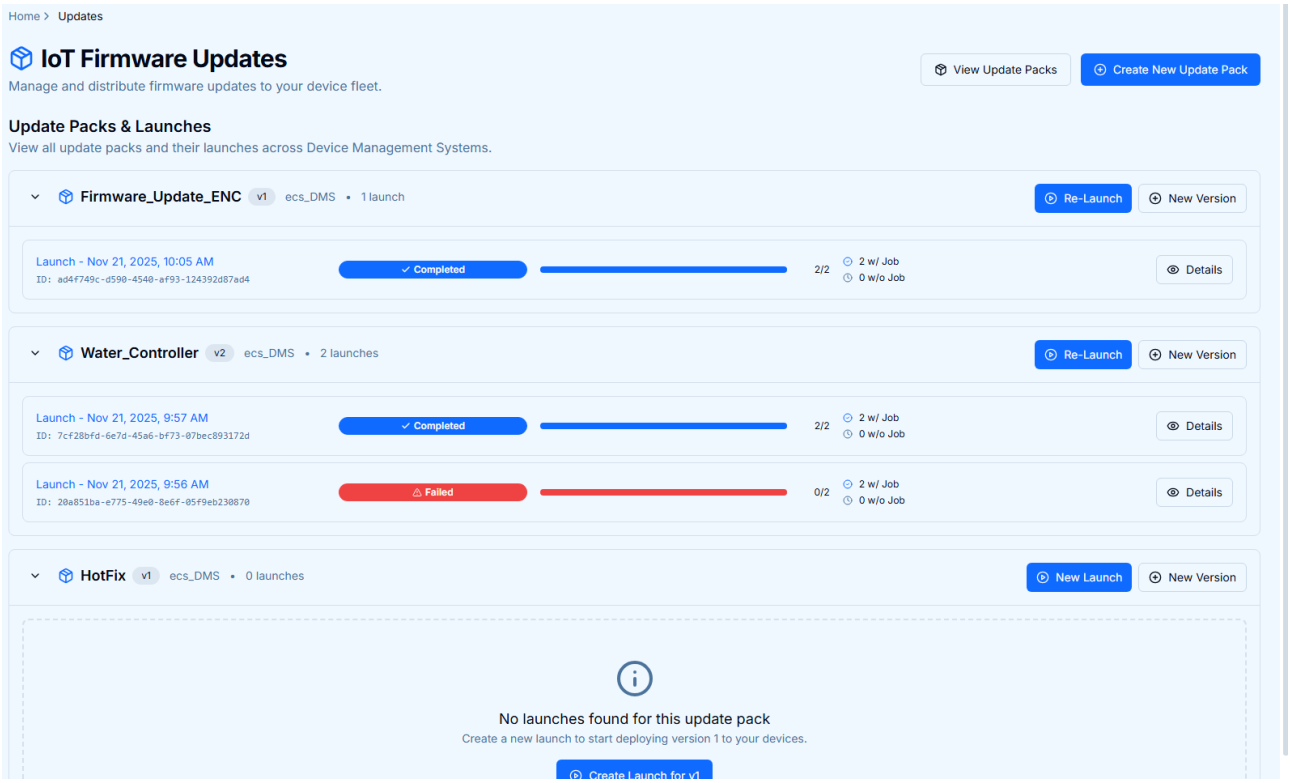

Figura 3.5: *Página principal de gestión de actualizaciones*

3.2.2.2. Creación de una Nueva Actualización

El formulario de creación de actualizaciones (ver Figura 3.6) guía al usuario a través del proceso de construcción de un paquete SWU. Los pasos incluyen:

1. **Subida de ficheros:** Carga de los binarios, scripts o archivos que componen la actualización.
2. **Descriptor de actualización:** Especificación del archivo `sw-description` que define la estructura y el proceso de instalación.
3. **Selección de claves:** Elección de las claves criptográficas para cifrado y firma.
4. **Configuración de cifrado:** Decisión sobre qué ficheros específicos deben cifrarse y cuáles pueden permanecer en texto plano.

Este proceso abstrae la complejidad de invocar SWUGenerator directamente, proporcionando una interfaz intuitiva que valida las entradas y genera automáticamente el paquete SWU con las configuraciones criptográficas seleccionadas.

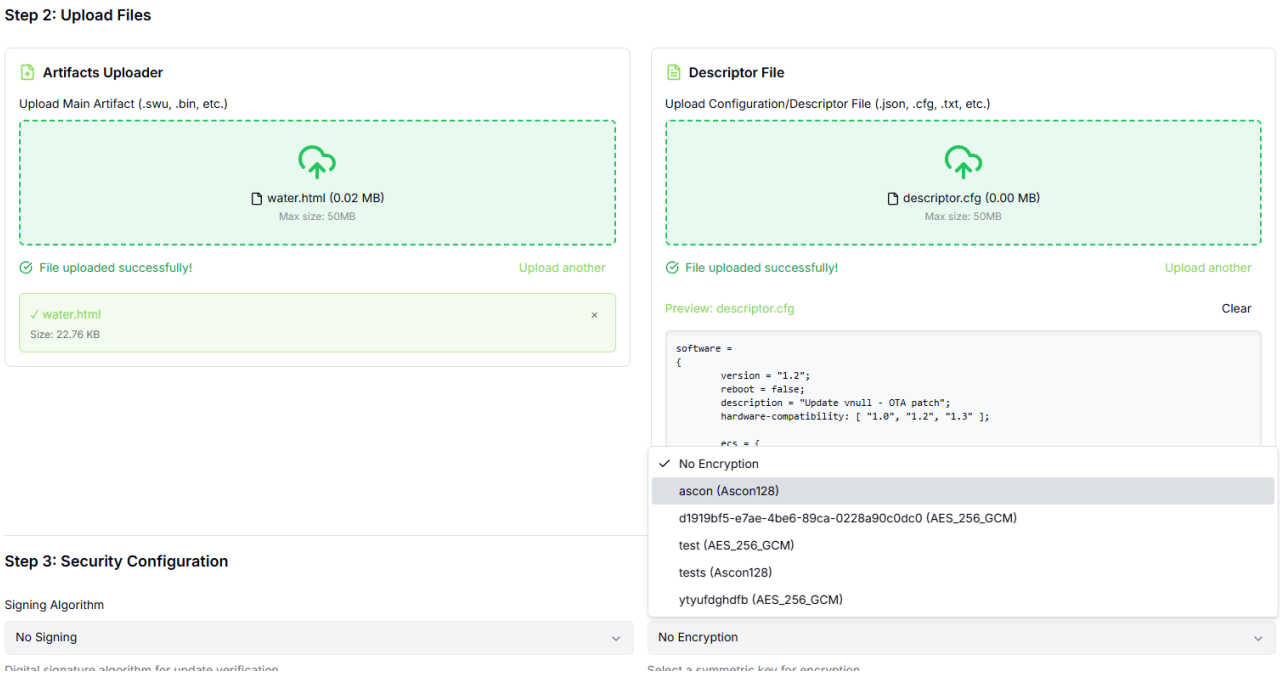

Figura 3.6: *Formulario de creación de una nueva actualización*

3.2.2.3. Detalles de un Update Pack

Al hacer clic en un paquete de actualización, se accede a una vista detallada (ver Figura 3.7) que muestra:

- **Metadatos completos:** Nombre, versión, tipo (firmware/file), algoritmos utilizados, etc.
- **Historial de versiones:** Todas las versiones creadas para este paquete.
- **Descarga del paquete:** Posibilidad de descargar el archivo `.swu` generado para inspección o instalación manual.

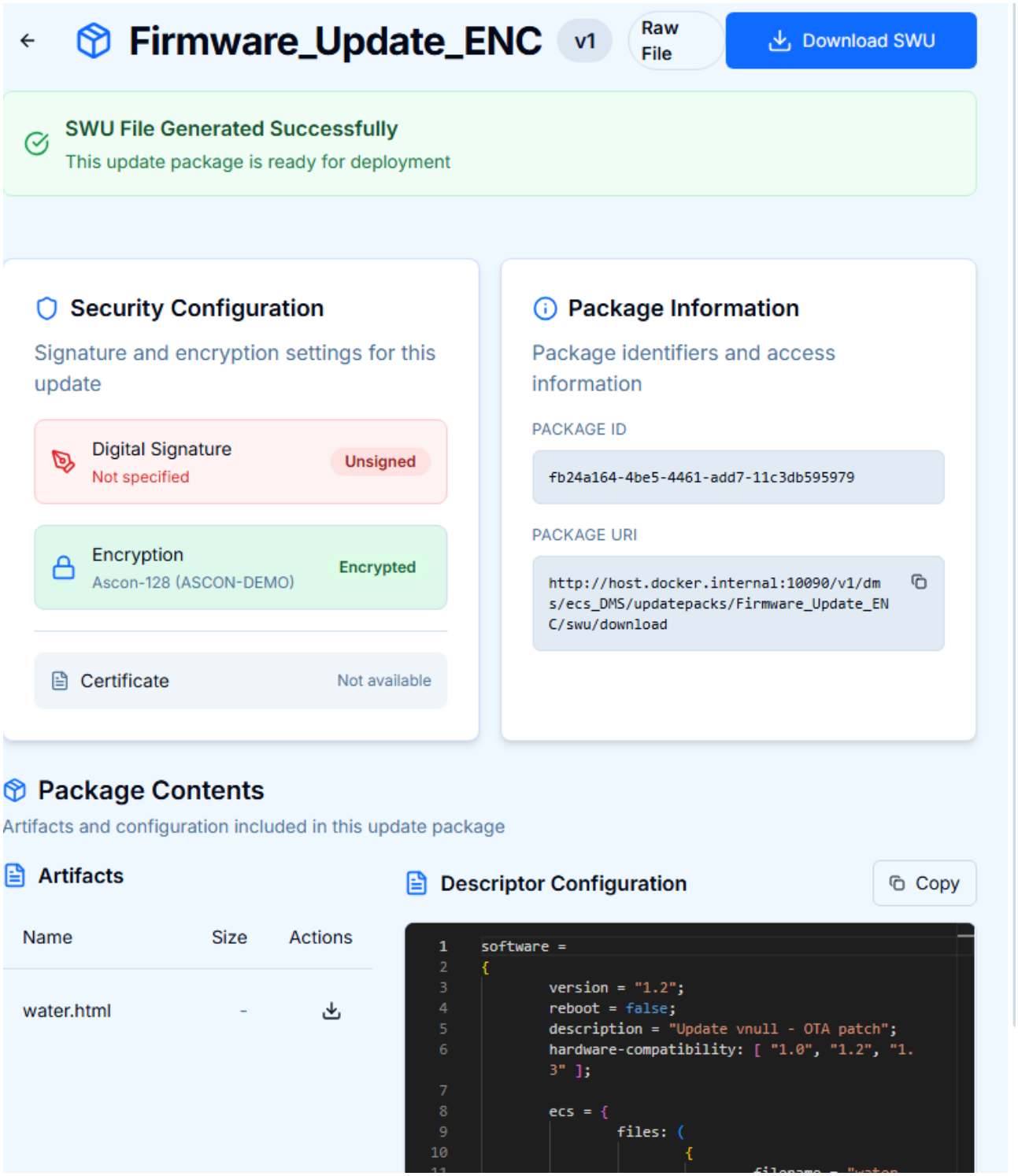

Figura 3.7: *Vista detallada de un paquete de actualización*

3.2.3 Gestión de Lanzamientos

3.2.3.1. Configuración de un Lanzamiento

Al iniciar un lanzamiento, se presenta un formulario de configuración (ver Figura 3.8) donde se especifican los parámetros de la campaña:

- **Tipo de workflow:** Selección entre Direct, Phased o workflows personalizados.
- **Estrategia de rollout:** Configuración de despliegue gradual por porcentaje o cantidad fija de dispositivos.
- **Cantidad**: Número o porcentaje de dispositivos a actualizar en cada fase (si aplica).
- **Paquete de actualización:** Selección del paquete de actualización que se desplegará.
- **Modo auto:** Activación o desactivación del modo automático para el despliegue de subgrupos.
- **Dispositivo de prueba:** Designación de un dispositivo piloto para validar la actualización antes del despliegue masivo.

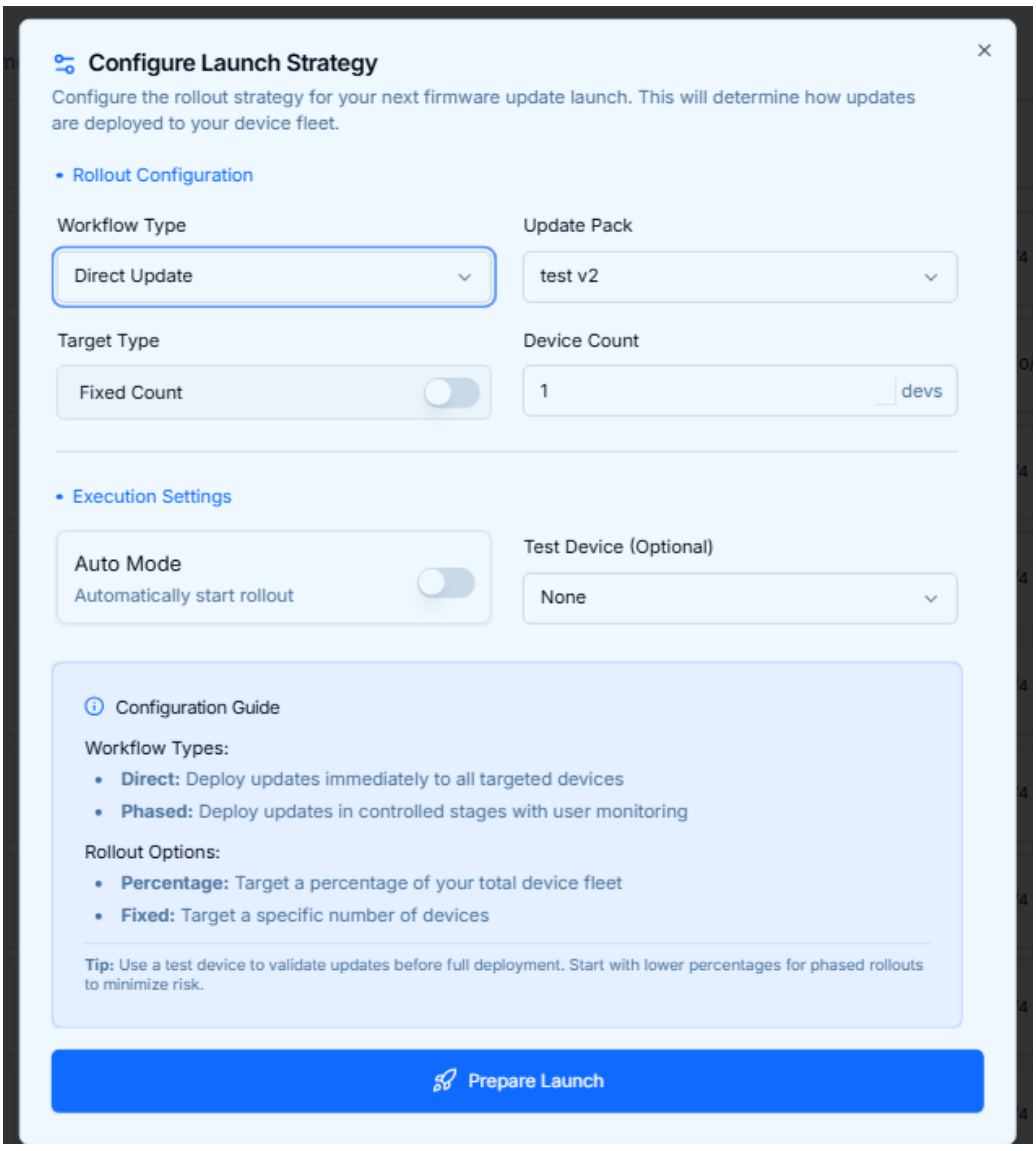

Figura 3.8: *Formulario de configuración de un lanzamiento*

3.2.3.2. Detalles de un Lanzamiento

La vista de detalles de un lanzamiento (ver Figura 3.9) proporciona visibilidad completa sobre el estado de la campaña:

- **Estado de cada dispositivo:** Lista de todos los dispositivos objetivo con su estado actual (pendiente, descargando, instalando, completado, error).
- **Estadísticas adicionales:** Número de dispositivos completados, activos y sin iniciar.
- **Control de flujo (workflows por fases):** En caso de utilizar un workflow Phased, se proporciona control manual para decidir qué dispositivos avanzan a la siguiente fase, o aprobar el avance de todos los dispositivos elegibles simultáneamente.

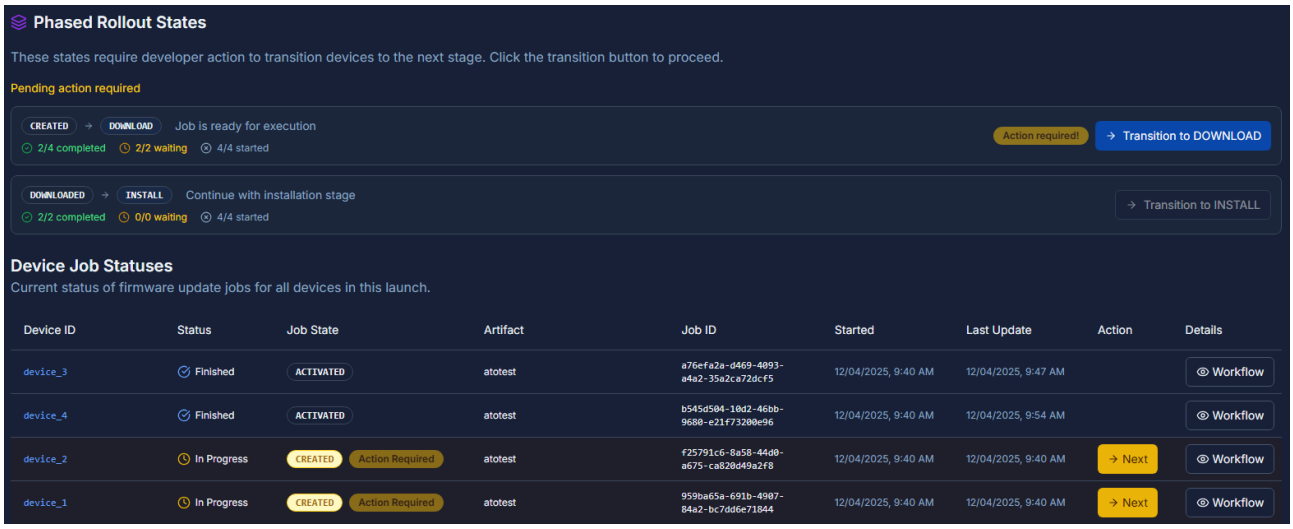

Figura 3.9: *Vista detallada de un lanzamiento con estado de dispositivos*

3.2.4 Monitorización de Dispositivos Individuales

3.2.4.1. Línea Temporal de Estados

Para cada dispositivo involucrado en un lanzamiento, se proporciona una visualización temporal del proceso de actualización (ver Figura 3.10). Esta interfaz presenta:

- **Diagrama de estados:** Representación gráfica del workflow utilizado, mostrando todos los estados posibles.
- **Estados completados:** Indicación visual de los estados por los que ha pasado el dispositivo.
- **Estado actual:** Resaltado del estado en el que se encuentra actualmente el dispositivo.
- **Información detallada:** Detalles adicionales de cada estado al pasar el cursor sobre él.

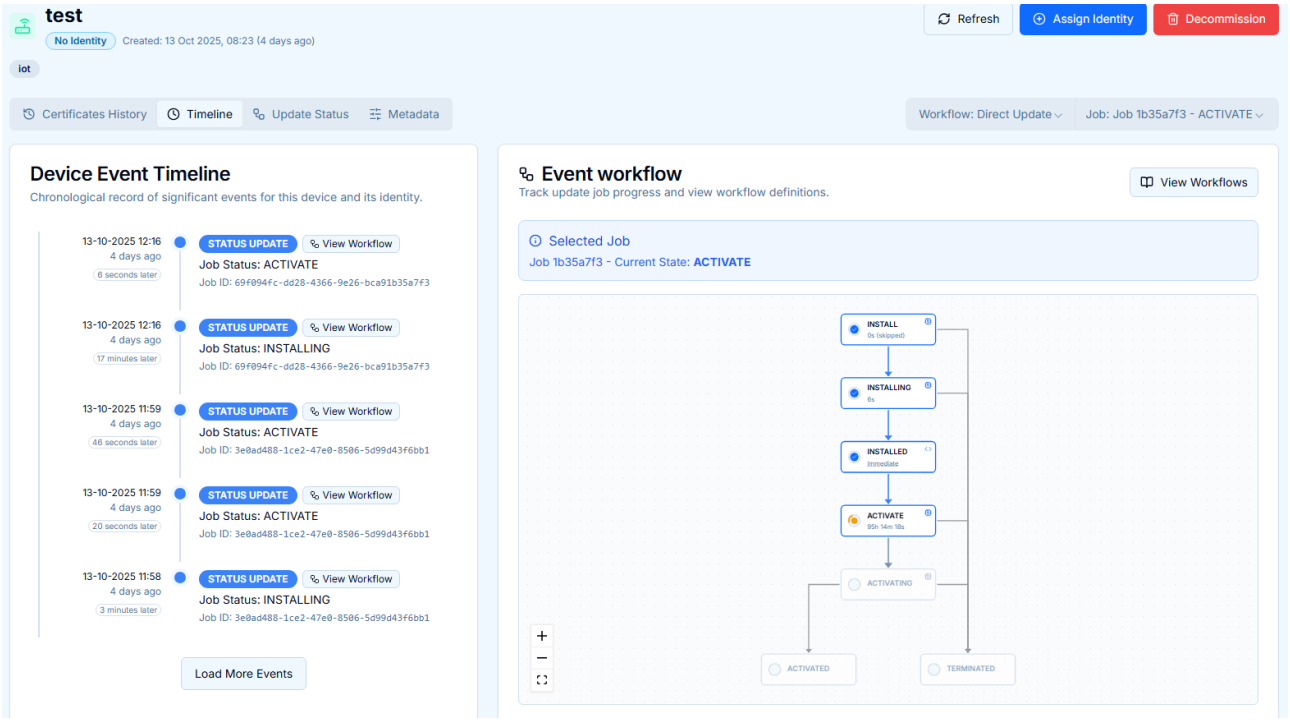

Figura 3.10: *Línea temporal de estados de instalación de un dispositivo*

3.2.4.2. Detalles de Estados Específicos

Al interactuar con un estado en la línea temporal, se despliega información contextual (ver Figura 3.11):

- **Tiempos de permanencia:** Duración que el dispositivo permaneció en ese estado.
- **Timestamp de entrada y salida:** Momentos exactos de las transiciones.

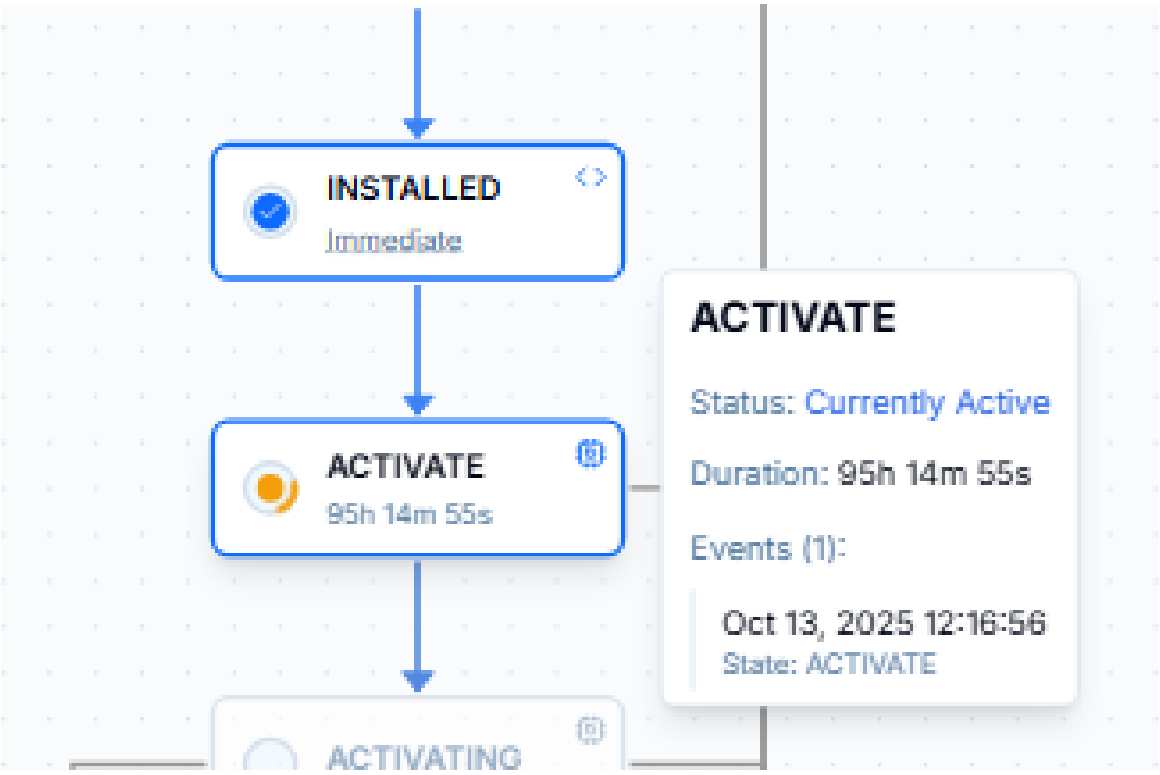

Figura 3.11: *Detalles temporales de un estado específico*

En caso de errores durante el proceso de actualización, la interfaz proporciona información detallada del fallo (ver Figura 3.12):

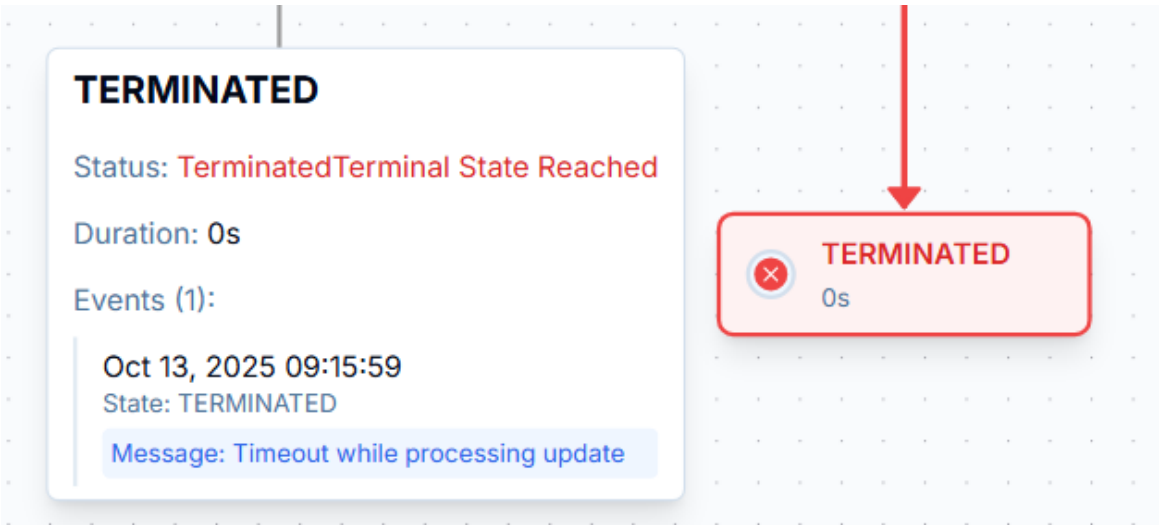

Figura 3.12: *Visualización de detalles de error en un estado*

Con esto se prueba la arquitectura propuesta con los cambios a las herramientas definidas, integrando criptografía ligera y post-cuántica en toda la cadena de actualización (generación, distribución e instalación) en una prueba de concepto funcional.

3.3 Resultados de Pruebas y Cobertura de Código

Como parte de la validación del sistema, se han ejecutado las baterías de tests descritas en la sección de desarrollo. A continuación se presentan los resultados obtenidos.

3.3.1 Cobertura de Código

La ejecución completa de los tests arroja los siguientes porcentajes de cobertura:

- **SymKMS:** 83.0 % de cobertura de código, con más de 150 tests ejecutados.
- **Updates:** 74.4 % de cobertura de código, con más de 70 tests ejecutados.

Las Figura 3.13 y Figura 3.14 muestran los *treemaps* de cobertura de ambos servicios, donde el tamaño de cada bloque es proporcional al volumen de código del paquete y el color indica el nivel de cobertura alcanzado (verde para alta cobertura, rojo para baja).

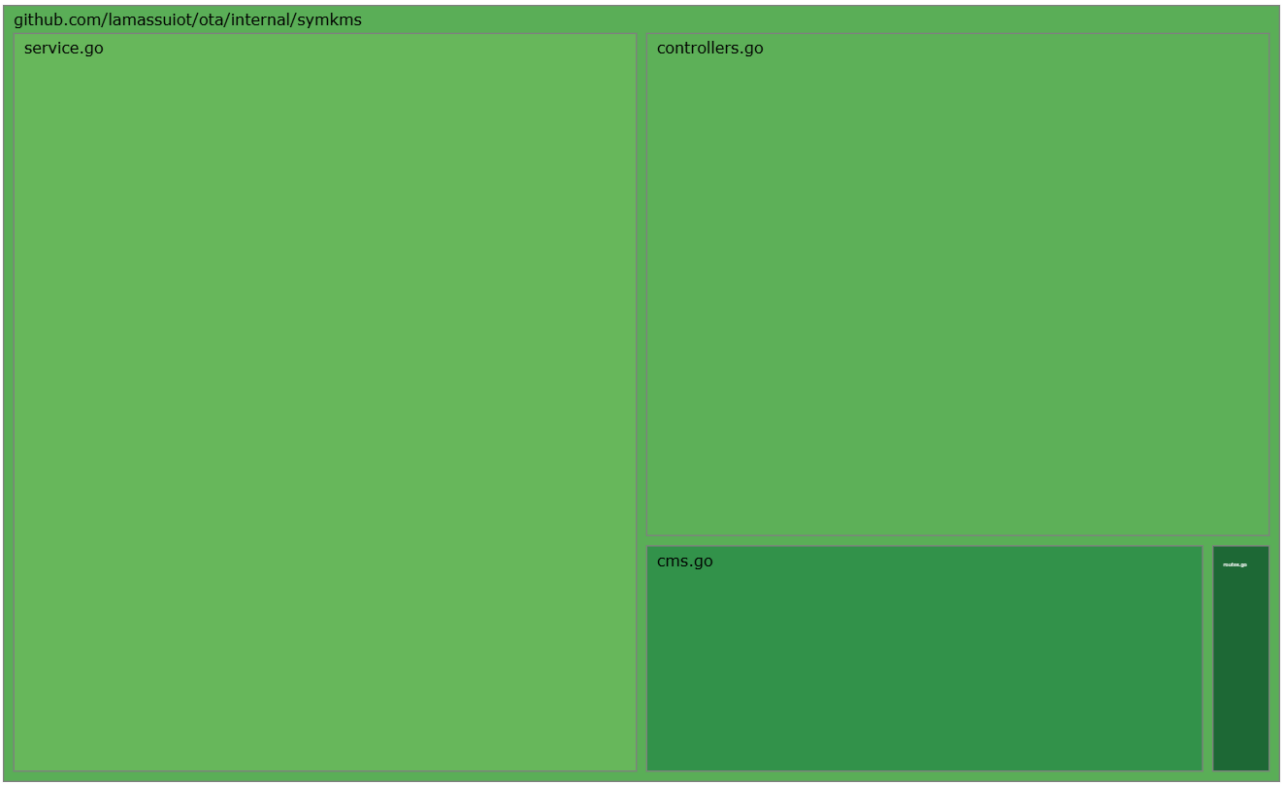

Figura 3.13: *Treemap de cobertura de código del servicio SymKMS (83.0 %)*

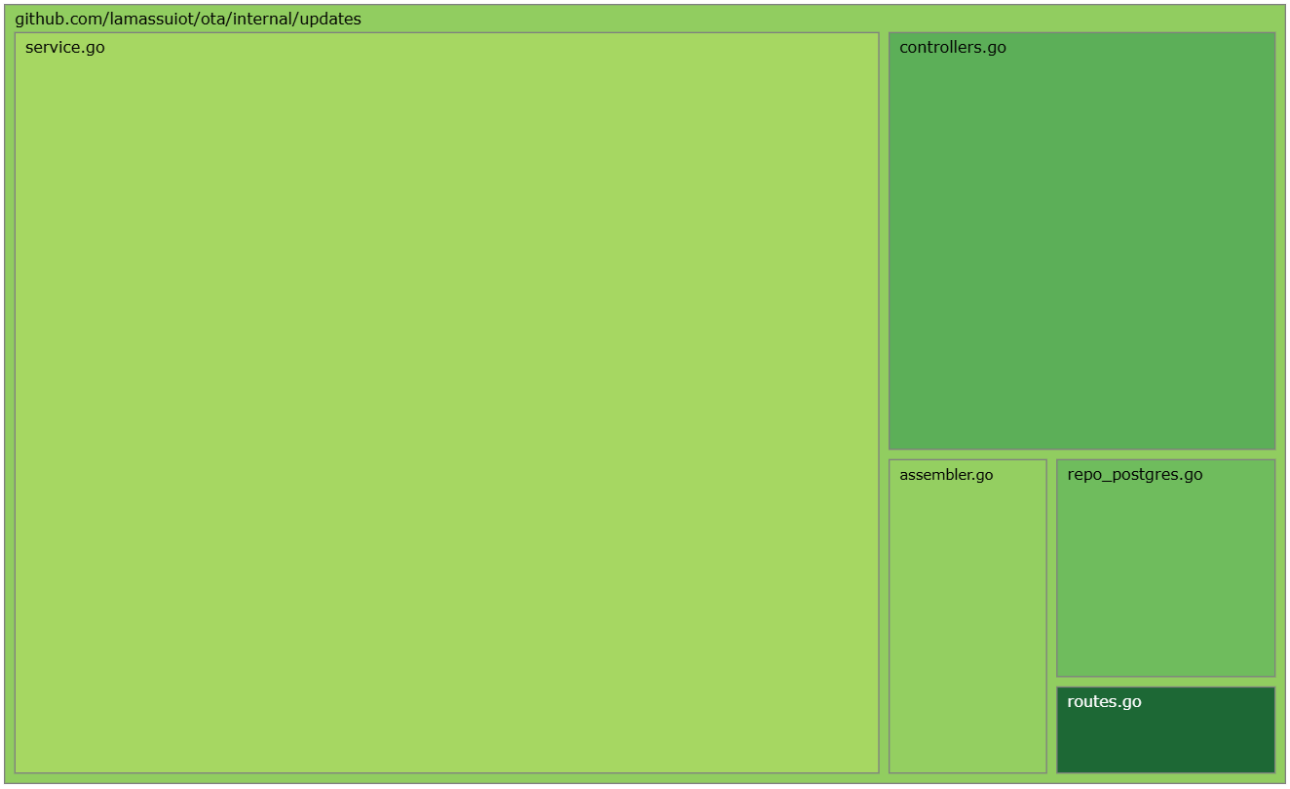

Figura 3.14: *Treemap de cobertura de código del servicio Updates (74.4 %)*

3.3.2 Validación Criptográfica

Los resultados de la validación criptográfica confirman la correctitud de las implementaciones:

- **AES-CMAC (RFC 4493):** Los 4 vectores de test oficiales pasan correctamente tras la validación cruzada con la implementación de referencia en Python. Se documentaron 2 errores tipográficos en la propia RFC, confirmados mediante la librería `cryptography` de Python.
- **ASCON:** Las 3 variantes (Ascon-128, Ascon-128a, Ascon-80pq) superan todas las pruebas de *round-trip*, detección de corrupción y validación con datos de longitud variable (desde vacío hasta 10 KB).
- **Interoperabilidad CMS/PKCS#7 con OpenSSL:** Se ha verificado la compatibilidad bidireccional: cifrado con la implementación propia y descifrado con OpenSSL, y viceversa. Todas las estructuras ASN.1 y OIDs (tanto estándar como personalizados para ASCON) son correctamente interpretados por ambas partes.

3.3.3 Tests de Integración

Los tests de integración, ejecutados contra contenedores Docker reales de PostgreSQL y WFX, validan el funcionamiento de la plataforma en condiciones similares a producción:

- **Persistencia:** Todas las operaciones CRUD sobre *UpdatePacks* y *LaunchTracks* funcionan correctamente, incluyendo consultas con paginación, filtrado y ordenación.
- **Concurrencia:** Las pruebas de 10 inserciones paralelas se completan sin conflictos ni pérdida de datos.
- **Orquestación WFX:** La carga de *workflows*, creación de *jobs* y gestión de estrategias de lanzamiento (directas y por fases) operan correctamente a través de la API de WFX.

Estos resultados confirman que los componentes desarrollados cumplen con los requisitos de corrección criptográfica, interoperabilidad con herramientas estándar y fiabilidad en entornos de integración realistas.

4. Conclusiones

Este trabajo ha abordado el desafío de desarrollar una plataforma segura y escalable para la distribución de actualizaciones OTA en dispositivos IoT, incorporando algoritmos de criptografía ligera (LWC) y resistentes a ataques cuánticos (PQ). Los resultados obtenidos demuestran que los objetivos planteados se han alcanzado de manera satisfactoria.

4.1 Cumplimiento de Objetivos

Se ha desarrollado con éxito una plataforma completa que permite distribuir actualizaciones de manera segura mediante cifrado y firma digital. La plataforma integra:

- **Soporte para criptografía ligera:** Integración de ASCON como alternativa eficiente a AES para dispositivos con recursos limitados.
- **Criptografía post-cuántica:** Incorporación de algoritmos de firma digital resistentes a ataques cuánticos.
- **Interfaz de gestión:** Desarrollo de una interfaz web intuitiva que abstrae la complejidad del proceso de actualización.

4.2 Validación de Criptografía Ligera en Dispositivos Limitados

Uno de los hallazgos más relevantes es la demostración práctica de que la criptografía ligera merece la pena en dispositivos con recursos limitados. Los algoritmos LWC, específicamente ASCON, han probado ofrecer:

- Menor consumo de recursos computacionales y de memoria.
- Rendimiento superior en dispositivos embebidos cuando se utiliza la implementación en C nativo.
- Seguridad equivalente a los estándares tradicionales.

Esta validación resulta especialmente importante para el ecosistema IoT, donde millones de dispositivos operan con recursos estrictamente limitados. La posibilidad de proporcionar seguridad robusta sin comprometer la viabilidad técnica representa un avance significativo.

4.3 Preparación para Amenazas Post-Cuánticas

La integración de algoritmos post-cuánticos constituye un cambio que merece la pena de cara al futuro. Aunque las computadoras cuánticas no son aún una amenaza práctica inmediata, la preparación anticipada es esencial considerando que:

- Los dispositivos IoT tienen ciclos de vida prolongados (10-20 años).
- Los adversarios pueden interceptar comunicaciones cifradas hoy para descifrarlas en el futuro.
- La migración a criptografía post-cuántica requiere tiempo de maduración y estandarización.

La plataforma proporciona la flexibilidad necesaria para soportar tanto algoritmos tradicionales como post-cuánticos, permitiendo una transición gradual según las necesidades específicas de cada despliegue.

4.4 Reflexión Final

La plataforma desarrollada demuestra que es posible construir infraestructuras de actualización seguras para dispositivos IoT sin comprometer la viabilidad técnica. La integración de criptografía ligera permite proteger dispositivos con recursos limitados, mientras que la incorporación de algoritmos post-cuánticos asegura la relevancia a largo plazo de la solución.

Al proporcionar una solución completa, usable y escalable, se facilita que organizaciones puedan implementar actualizaciones OTA seguras mediante cifrado y firma digital, preparadas para las amenazas del futuro.

Bibliografía

[1] Alison Gonçalves Schemitt, Henrique Fan da Silva, Roben Castagna Lunardi, Diego Kreutz, Rodrigo Brandão Mansilha, and Avelino Francisco Zorzo. Assessing the impact of post-quantum digital signature algorithms on blockchains, 2025.

[2] Gartner. IoT security surveys and reports. Technical report, Gartner, 2023. Consultado en 2023.

[3] Kaspersky Security. Mirai Botnet and IoT-Based DDoS Attacks: Evolution and Current Threats. https://www.kaspersky.com/resource-center/threats/mirai-botnet, 2025. Análisis de botnets IoT y ataques DDoS masivos impulsados por dispositivos comprometidos.

[4] Muhammad Ibrahim, Andrea Continella, and Antonio Bianchi. AoT - Attack on Things: A security analysis of IoT firmware updates. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 1047–1064, 2023.

[5] Yewande Goodness Hassan, Anuoluwapo Collins, Gideon Opeyemi Babatunde, Abidemi Adeleye Alabi, and Sikirat Damilola Mustapha. Security Challenges in Industrial IoT: Update Mechanisms and Firmware Vulnerabilities. *International Journal/Conference Name*, 04(01):697–703, January-February 2023. Received: 12-12-2022; Accepted: 19-01-2023.

[6] Cyber resilience act, 2024. Regulation (EU) 2024/1689.

[7] Iec 62443-3-3: Industrial communication networks - network and system security - part 3-3: System security requirements and security levels. Technical report, International Electrotechnical Commission (IEC), 2021. Standard for industrial automation and control systems security.

[8] Cisco Systems. Cisco Annual Internet Report (2018–2023) and IoT Forecast. Technical report, Cisco Systems, 2025. Proyección de más de 29 mil millones de dispositivos IoT conectados para 2025.

[9] Fortinet. Top 10 IoT Vulnerabilities and Security Threats. https://www.fortinet.com/resources/cyberglossary/iot-security, 2024. Informe sobre las principales vulnerabilidades en dispositivos IoT, incluyendo mecanismos de actualización inseguros y firmware obsoleto.

[10] Bitdefender Labs. Critical Vulnerabilities in ThroughTek Kalay Platform Affect Over 100 Million IoT Devices. Technical report, Bitdefender, May 2024. Vulnerabilidades críticas en la plataforma TUTK que afectan a más de 100 millones de dispositivos IoT globalmente.

[11] SWUpdate Project. SWUpdate - Software Update for Embedded Linux Devices, 2024. Framework de código abierto para actualizaciones de firmware en sistemas Linux embebidos con soporte para actualizaciones atómicas A/B y firmas RSA/ECDSA.

[12] Eclipse Foundation. Eclipse hawkBit - IoT Update Management, 2024. Plataforma backend de código abierto para gestión de actualizaciones OTA a escala empresarial con rollout progresivo y gestión de campañas.

[13] Northern.tech. Mender - Over-the-Air Software Updates for IoT, 2024. Plataforma OTA que integra cliente y servidor para actualizaciones con gestión de flotas, rollback automático e interfaz gráfica.

[14] Balena Inc. Balena - IoT Fleet Management Platform, 2024. Plataforma de gestión de flotas IoT basada en contenedores Docker para despliegue y actualización de aplicaciones.

[15] RAUC Project. RAUC - Robust Auto-Update Controller, 2024. Framework de código abierto para actualizaciones seguras de sistemas Linux embebidos con soporte para múltiples esquemas de particionado.

[16] National Institute of Standards and Technology (NIST). Lightweight Cryptography: Program Document. Special Publication (SP) 800-232, U.S. Department of Commerce, 2023.

[17] National Institute of Standards and Technology (NIST). Ascon-Based Lightweight Cryptography Standards for Constrained Devices. Special Publication (SP) 800-232, U.S. Department of Commerce, 2025. Estándar oficial de criptografía ligera del NIST basado en ASCON para dispositivos con recursos limitados.

[18] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Status Update on Ascon v1.2. Status update, NIST Lightweight Cryptography Standardization Process, September 2022. Finalist Round Status Update con benchmarks de rendimiento en dispositivos con recursos limitados.

[19] Raspberry Pi Foundation. Raspberry Pi, 2025. Sitio web oficial de Raspberry Pi.

[20] Arduino. Arduino IDE, 2025. Entorno de desarrollo integrado oficial para Arduino.

[21] Christoph Dobraunig and others. ASCON C Implementation, 2023. Implementación oficial de ASCON recomendada por NIST.

[22] U.S. Department of Commerce. National Institute of Standards and Technology (NIST), 2025. Sitio web oficial del NIST.

[23] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[24] National Institute of Standards and Technology (NIST). Module-Lattice-Based Digital Signature Standard, 2024.

[25] Yocto Project. System Update. https://wiki.yoctoproject.org/wiki/System_Update, 2024. Accedido: 12-12-2025.

[26] Wfx. https://siemens.github.io/wfx/. Accedido: 12-12-2025.

[27] Stefano Babic. SWUGenerator: Tool to generate SWUpdate update packages. https://github.com/sbabic/swugenerator, 2024. Accedido: 12-12-2025.

[28] ascon: Ascon v1.2 authenticated encryption and hashing. https://pypi.org/project/ascon/. Accessed: 2026-02-12.

[29] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4. In *Second PQC Standardization Conference*, 2019. Updated results available at https://github.com/mupq/pqm4.