



INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY

ESCUELA DE CIENCIAS APLICADAS

**Uso de álgebras modernas para
seguridad y criptografía**

Grupo: 01

**Reporte Final
Unidad MA2006B**

25 DE ABRIL DE 2023

Profesores:

Dr. Alberto F. Martinez

Dr. Salvador Mancilla

Organización Socioformadora:

LiCore

Alumnos:

Pablo Ramirez Santaella Urencio A01197501

Francisco Leonid Gálvez Flores A01174385

Francisco Castorena Salazar A00827756

Daniel Sánchez Villarreal A01197699

Iker Ledesma Durán A01653115

Índice

1. Introducción	4
2. Estado del Arte	5
2.1. Firma Digital	6
2.2. ECDSA	7
2.3. Casos Aplicados de uso de algoritmos criptográficos y dispositivos tipo SoC en ambientes IoT	7
2.4. AES-128	8
2.5. Arquitecturas de intercambio de información en red	9
2.6. Recursos hardware disponibles	12
2.7. Tabla Comparativa de Componentes	12
3. Propuesta	13
3.1. ECC (Elliptic-Curve Cryptography)	15
3.2. ECDSA (Elliptic Curve Digital Signature Algorithm)	15
3.3. SHA-256	16
4. Metodología y Desarrollo	21
4.1. Transformación de Datos a json	22
4.2. Montaje de Base de Datos y Almacenamiento de Datos	23
4.3. Montaje de Segunda Base de Datos	23
4.3.1. Implementación de MicroPython en ESP32	24
4.4. Costos Firebase	24
4.4.1. Aproximado de costos para la situación problema	26
4.5. Conexión de ESP32 a Base de Datos y Centro de Control	27
4.6. Esquema Criptográfico, implementación de SHA-256 y ECDSA	28
4.6.1. Librerías y recursos a utilizar	28
4.6.2. SHA-256	29
4.6.3. ECDSA	29
4.6.4. Algoritmo de Verificación de Firma	31
4.7. Implementación ECDSA MicroPython	32
4.8. Montado del Centro de Control	34
4.8.1. Conexión entre ESP32 y Arduino Cloud	36
4.9. Creación de Certificados	38
5. Vectores de Prueba	39
6. Resultados	41
7. Conclusiones	42

7.1. Recomendaciones	43
--------------------------------	----

Resumen

Este estudio examina arquitecturas y algoritmos criptográficos para proteger datos en Internet, especialmente en ambientes IoT, se busca enviar datos de medición de consumo y producción de energía desde un auditor a un centro de control de forma que se puede garantizar la integridad, confidencialidad y autenticidad de estos. Se propone una solución efectiva basada en una metodología sólida y resultados concretos. Se emplea una arquitectura de red con un servidor que contiene los datos en crudo en un archivo json y se accede a ellos mediante MicroPython y un chip SoC tipo ESP32. Luego, se encriptan los datos con ECDSA y SHA-256 y se envían a un servidor destinatario. El centro de control verifica la autenticidad de los mensajes con una clave pública. La solución propuesta ha demostrado ser efectiva para el cifrado y verificación de los datos, donde en nuestro caso el tiempo promedio requerido para recibir un dato, cifrarlo, enviarlo y verificarlo por el centro de control es de aproximadamente 15.2 segundos. Se concluye con recomendaciones para futuros estudios con el propósito de mejorar el sistema implementado.

1. Introducción

En este reto, la seguridad será un factor clave y se utilizará la criptografía de clave pública para proteger las comunicaciones WiFi entre dispositivos IoT y un centro de monitoreo de consumo de energía. La existencia de riesgos y amenazas en cuanto a robo de datos digitales es una preocupación en canales de comunicación que no cuentan con medidas de protección adecuadas. Por esta razón, se buscará implementar esquemas de clave pública que funcionen en entornos con recursos limitados.

El propósito principal de este proyecto es asegurar la comunicación segura entre todas las partes involucradas en la red, incluyendo sensores, centro de control y auditor. Escalabilidad, código fuente y documentación valiosa serán compartidos en plataformas como Github, siempre teniendo en cuenta la confidencialidad de los datos sensibles compartidos por el socio formador o generados por parte del proyecto. Este reto estará trabajado en conjunto con la organización LiCore, la cual actuará como socio formador. El objetivo principal será la implementación de protocolos de criptografía de clave pública para proteger la información durante los intercambios de datos entre sensores en un entorno IoT. Se buscará que estos intercambios sean seguros y automatizados, incluyendo información sobre el estado de los sensores y el monitoreo y consumo de energía.

Asimismo, proteger el intercambio de datos entre el auditor y el centro de control será una de las misiones clave de este proyecto. Se buscará garantizar la confidencialidad, integridad y autenticidad de la información intercambiada. La confidencialidad se logrará asegurándose de que solamente las partes autorizadas tengan acceso a la información. La integridad se asegurará mediante la verificación de que los datos no sean modificados durante el intercambio. La autenticidad se logrará monitoreando y identificando cada sensor para determinar la origen de los datos. Es importante destacar que los intercambios de datos pueden incluir información sensible, por lo que la privacidad, confidencialidad y seguridad de los datos es fundamental. Además, el centro de control deberá ser capaz de actualizar el software del auditor dentro del esquema de intercambio de información que se

está protegiendo, garantizando que la solución sea actualizada y cumpla con las mejores prácticas de seguridad.

En resumen, este proyecto buscará garantizar que la información sea procesada de forma eficiente y segura, lo que permitirá a los usuarios obtener resultados precisos y confiables en un tiempo razonable. Además, se espera que la implementación de tecnologías avanzadas en el procesamiento de datos permita una mayor eficiencia en la toma de decisiones y una mejora en la calidad de los servicios prestados a los usuarios. En general, se espera que este proyecto tenga un impacto positivo en la industria y contribuya a la innovación en el procesamiento de datos.

Tomando en cuenta lo anterior, se buscará imitar un entorno de comunicaciones, similar al de la siguiente imagen. (Imagen proporcionada por recursos de clase en la plataforma de Canvas, [31] disponible en <https://experiencia21.tec.mx/courses/344009/pages/reto>):

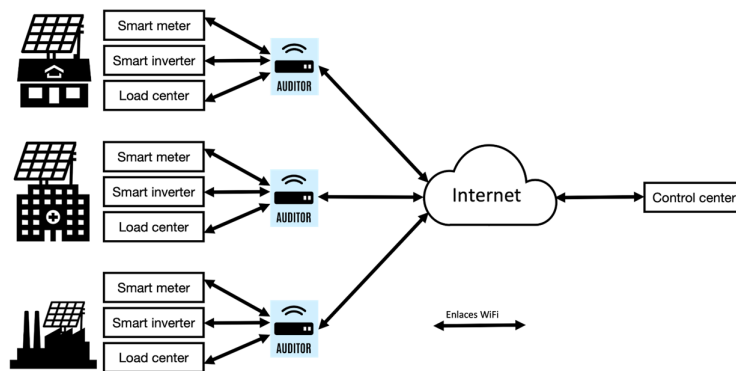


Figura 1: Esquema de intercambio de datos a ser protegido. (Imagen proporcionada por recursos de clase en la plataforma de Canvas, [31] disponible en <https://experiencia21.tec.mx/courses/344009/pages/reto>)

Para apegarse a este esquema mostrado en la Figura 1, se requerirá que el auditor pueda funcionar con recursos limitados como se mencionó anteriormente. Un ejemplo de esto puede ser con respecto al consumo de memoria. De igual manera, se buscará que el auditor tenga el propósito de recolectar, encriptar y mandar los datos recibidos desde los dispositivos de medición de energía hacia un centro de control.

2. Estado del Arte

Cuando se utiliza el internet como medio de comunicación, se deben de aplicar algunas medidas de seguridad pertinentes, estas pueden ser sockets, conexiones VPN, etc.[29] Normalmente se usan

protocolos SSL, el cual es un estándar en la protección de comunicaciones a nivel cliente servidor; retomando la red VPN, esta sirve para establecer un canal de comunicación completamente seguro entre dos puntos finales, este al igual que el protocolo anterior están cifrados, pero la VPN permite encapsular el flujo de tráfico, esto para así proporcionar un mejor nivel de seguridad a la red y con esto hace que nadie ajeno a dicho intercambio de información conozca el contenido de la misma. Ahora bien, una vez tomando en cuenta que se está en un ámbito de IoT, se debe recordar que los dispositivos IoT son más susceptibles a vulnerabilidades de firmware, debido a que no poseen una gran cantidad de protecciones de seguridad, además de que estos con frecuencia no pueden ser actualizados, lo que los deja completamente expuestos a una gran cantidad de ataques, los cuales pueden ser[29]:

- Ataques basados en credenciales.
- Ataques en ruta.
- Ataques basados en el hardware físico.

Algunas formas de mantener la seguridad en estos dispositivos pueden ser la actualización del firmware de estos, mantener la seguridad en sus credenciales (no reutilizar), autenticación de dispositivos entre sí mediante certificados TLS, encriptación de los envíos de información, desactivar vulnerabilidades apagando funciones que no se utilicen, uso de filtros DNS, entre otras. [29]

2.1. Firma Digital

Las firmas digitales emplean criptografía asimétrica. En muchos casos, brindan una capa de validación y seguridad a los mensajes enviados a través de un canal no seguro: correctamente implementada, una firma digital le da al receptor razones para creer que el mensaje fue enviado por el supuesto remitente. Las firmas digitales son equivalentes a las firmas manuscritas tradicionales en muchos aspectos, pero las firmas digitales implementadas correctamente son más difíciles de falsificar que las firmas manuscritas. Los esquemas de firma digital, en el sentido utilizado aquí, se basan en criptografía y deben implementarse correctamente para que sean efectivos. También pueden proporcionar un factor conocido como no repudio, lo que significa que el firmante no puede afirmar con éxito que no firmó un mensaje, al mismo tiempo que afirma que su clave privada permanece en secreto. Además, algunos esquemas de no repudio ofrecen una marca de tiempo para la firma digital, de modo que incluso si se expone la clave privada, la firma es válida. Los mensajes firmados digitalmente pueden ser cualquier cosa que pueda ser representada como una cadena de bits: los ejemplos incluyen correo electrónico, contratos, o un mensaje enviado a través de algún otro protocolo criptográfico. [37]

2.2. ECDSA

El algoritmo de firma digital de curvas elípticas (conocido como ECDSA por sus siglas en inglés), al igual que con la criptografía de curva elíptica en general, el tamaño en bits de la clave privada que se cree necesaria para ECDSA es aproximadamente el doble del tamaño del nivel de seguridad, en bits. Por ejemplo, con un nivel de seguridad de 80 bits, lo que significa que un ciberatacante requiere un máximo de aproximadamente 2^{80} operaciones para encontrar la clave privada, el tamaño de una clave privada ECDSA sería de 160 bits. Por otro lado, el tamaño de firma es el mismo tanto para DSA como para ECDSA: aproximadamente $4t$ bits, donde t es el nivel de seguridad medido en bits, es decir, unos 320 bits para un nivel de seguridad de 80 bits. Mayores detalles se verán en la sección de Metodología y Desarrollo. [47]

2.3. Casos Aplicados de uso de algoritmos criptográficos y dispositivos tipo SoC en ambientes IoT

Investigando respecto a casos en general donde se buscaba que el intercambio de datos entre dispositivos IoT fuera seguro, se encontró que en algunos de estos casos se hace uso de curvas elípticas para garantizar un ambiente de comunicación privado y seguro, pues este método tiene la capacidad de generar claves que puedan ser seguras pero que no sean muy largas, y además puede tener un funcionamiento muy eficiente. [36]. Asimismo, se encontraron otros casos donde se hace uso de otros algoritmos, como por ejemplo Blowfish, para monitorear y transmitir de forma segura distintos tipos de datos. Por otra parte, se encontró también que la implementación de chips con el ESP32 y el ESP8266, han llegado a ser utilizados en la práctica [21]. Ejemplos encontrados de este tipo de situaciones y prácticas mencionadas se muestran a continuación:

- Una de las primeras compañías en llevar esto a cabo fue la compañía AXEL ELECTRONICA, quienes lograron juntar una alta cantidad de sensores sobre el esquema de un Arduino M0+, entre esto un receptor GPS y un módulo de red inalámbrico SIGFOX junto con autenticación criptográfica para posteriormente sacar un nuevo modelo pensado específicamente para internet de las cosas, o como mejor se conoce, Internet of Things (IoT). Este modelo además tiene características como módulo de red WiFi y cripto-autenticador de curvas elípticas. [46]
- Otro ejemplo es la agencia EADTrust (European Agency of Digital Trust) [48], que presta sus servicios en Europa para cuestiones de seguridad digital. Entre estas cuestiones se encuentra el firmado digital [42] y ofrece certificaciones digitales mediante las cuales se basa en el uso de curvas elípticas para llevar esto a cabo. Además el certificado de ellos, solo pesa 2 kilo-bytes, lo que lo hace un buen prospecto para entornos de recursos limitados, poca memoria, dispositivos IoT, entre otros. [41]
- Uso del chip ESP32 para el monitoreo de estaciones de transmisión de energía en términos de eficiencia energética y diagnóstico de fallas, el desarrollo de aplicaciones desde un chip ESP32 puede ser hecho desde ambientes y lenguajes como Arduino Core y MicroPython. [17]

- Hoy en día, la demanda de comunicación segura a través de una red está creciendo de forma espectacular. Diferentes áreas como automoción, Internet de las cosas (IoT), sanidad, almacenamiento y los servicios financieros requieren el intercambio de información confidencial en canales inseguros. Sin embargo, en ocasiones uno puede tener la incertidumbre o duda de cuál es la mejor para utilizar y así mantenerse protegido. Para ello, en un estudio de la Universidad Técnica de Viena (UT Wien), realizado por los autores Clemens Lachner y Schahram Dustdar, se decidieron hacer comparaciones de desempeño de distintos equipos de hardware con la finalidad de encontrar cuál era más óptimo. En dicho estudio se llegó a la conclusión de que las más óptimas son ESP32 y ESP8266. [21]
- Otro ejemplo es el del algoritmo denominado como Blowfish, con el cual se puede hacer implementaciones para el cifrado de datos enviados desde una red IoT que tiene datos basados en una IP. Este se puede implementar para monitorear la cantidad de recursos usados. En un reporte publicado por los autores Prasetyo, Purwanto, y Darlis, disponible en [55], los autores analizan este algoritmo y calculan factores tales como la seguridad, tiempo de cifrado, rendimiento en varios escenarios para el grado de confiabilidad, entre otros. Aunque no emplea el uso de curvas elípticas, tras las pruebas que se hicieron, se obtuvo que este algoritmo tuvo un buen rendimiento al implementarse y mostró que podría ser una buena alternativa como seguridad de red en entorno IoT.
- Finalmente, otro ejemplo es el dispositivo Atmel ATECC508A que dentro de sus funcionalidades incluye el uso de criptografía de curva elíptica y protección de clave por hardware. Asimismo utiliza los protocolos de curva elíptica con Diffie-Hellman (ECDH) para intercambio de llaves [33]. De igual manera, a modo de ejemplo adicional, el dispositivo Atmel ATECC108A también incluye el uso de criptografía de curva elíptica.

2.4. AES-128

Asimismo, investigando respecto a distintos algoritmos de encriptación que son utilizados en dispositivos IoT, se encontró que el algoritmo AES-128 puede ser un buen prospecto para encriptar datos; esto debido a que es relativamente ligero, pues es menos pesado que el algoritmo AES-256 y con una llave de 128 bits. Un punto no favorable es que puede ser vulnerable a ataques del tipo Man-in-the-middle. [58]

Existen modificaciones al algoritmo AES original para hacerlo más ligero y rápido, y para una mejor integración en dispositivos IoT con bajo consumo de energía, como lo podría ser un algoritmo híbrido AES-ECC. Este además es flexible y versátil, con un diseño optimizado de la función ECC de acuerdo con las características de las redes de sensores inalámbricos. Con esto, incluso imágenes de textura podrían cifrarse utilizando procedimientos de enmascaramiento de bits y permutaciones. Con una terminal de interfaz de usuario en Python, el tiempo de cifrado puede ser aproximadamente un 50 por ciento más rápido que el AES normal. [22]

2.5. Arquitecturas de intercambio de información en red

Se definirá como una arquitectura de intercambio de información a aquel sistema compuesto por un conjunto de equipos de transmisión, programas, protocolos de comunicación y una infraestructura radioeléctrica que posibilite la conexión y transmisión de datos a través de una red. [24]

Existen distintos modelos de arquitectura de red de acuerdo al propósito de la red a implementar, se investigó respecto a aquellas que podrían ser útiles para la naturaleza de este reto, en donde, por una parte se desea que cada auditor (que en este caso, fungirá como un dispositivo del tipo IoT) se pueda comunicar de forma independiente con un centro de control, y, por otra parte, que el centro de control no esté conectado a la misma red que la de los auditores, esto para poder tener acceso de los datos de forma remota. Se encontraron las siguientes arquitecturas de red y se consideraron como útiles para esta situación problema:

- Peer-to-Peer Network (P2P):
 - Este tipo de arquitectura surgió a finales de los años setenta. Cada dispositivo conectado a la red actúa como un nodo con la capacidad y propósito de intercambiar archivos con otros nodos dentro de la red. A su vez, cada nodo actúa como servidor y no se tiene un servidor central en la red, teniendo así una gran capacidad de intercambio de datos en la red. La carga de trabajo de la red está dividida de forma equitativa entre sus nodos, y estos trabajan de forma independiente. [15]
 - Empresas como NABTO, dan uso de sistemas P2P para tener conexiones directas con dispositivos IoT, esto con el fin de tener bajos niveles de latencia entre dispositivos. [51]
 - También se han implementado sistemas de monitoreo de pacientes en hospital por medio de IoT y redes del tipo P2P. Un ejemplo de esto son las SmartBox vistas en la referencia [38]. Estos dispositivos cuentan con distintos sensores para el monitoreo del paciente y su entorno de forma exitosa.
 - Asimismo, con estas redes P2P se pueden usar también para crear programas, dominios o aplicaciones que sirven para distribuir de forma pública o privada documentos y archivos de tamaño considerable. Estos pueden estar en un dispositivo o dispersados entre varios dispositivos. Con esto, se pueden compartir copias de archivos o documentos hacia otros dispositivos y hacia prácticamente cualquier parte del mundo. Ejemplos de esto incluyen programas como eMule, iMesh, Ares, Azureus, entre otros. Aunque, como todo esto se puede utilizar con fines maliciosos también, pues este tipo de dominios, programas y aplicaciones son muy propensos a ser comprometidos con el fin de poner diversos tipos de virus, spam, malware y código malicioso para comprometer los dispositivos de usuarios que no tengan cultura de ciberseguridad o conocimientos o hábitos de cuidado con respecto a esta cuestión. [43]

- Ejemplo de conexiones en red P2P:

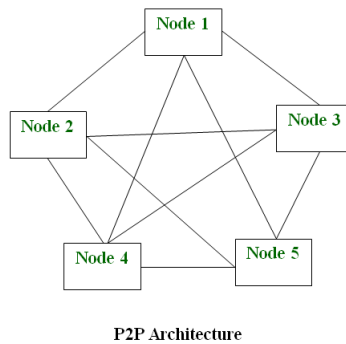


Figura 2: Conexiones en una red P2P. [15]

Esta Figura 2 ilustra como cualquier dispositivo conectado a la red P2P, tiene una conexión directa con los demás dispositivos.

- A su vez, existen distintos tipos de arquitectura de red del tipo P2P, estas son redes P2P estructuradas, no estructuradas e híbridas. [25] Estas se explican brevemente a continuación:
 - Redes P2P estructuradas: Los nodos de la red solo pueden interactuar con nodos específicos con el propósito de optimizar el intercambio de datos.
 - Redes P2P no estructuradas: Las conexiones entre nodos no están organizadas de ninguna forma, el proceso de comunicación entre nodos es aleatorio.
 - Redes P2P híbridas: Se combina la arquitectura con el modelo de cliente-servidor.
- Se considera que la red P2P puede ser de gran utilidad para el reto debido a que la independencia entre dispositivos permite que en caso de que un dispositivo falle en sus tareas o esté offline, no afecte al funcionamiento de la red, esto sería especialmente útil en redes donde se tuviera más de un solo auditor. Por otra parte, teóricamente, las redes P2P son más rápidas que las redes del tipo cliente-servidor, las cuales se explicarán a continuación.
- Modelos Cliente-Servidor: En este tipo de proceso, el intercambio de datos se realiza entre 2 dispositivos, estos pueden tener sistemas operativos distintos. Por una parte se tiene al cliente, el cual solicita información al servidor, mientras el servidor utiliza sus propios recursos para realizar los servicios solicitados por el cliente. [28] Este tipo de redes pueden ser esenciales para dispositivos IoT debido a que permite un monitoreo remoto de la red. Por otra parte, el

propósito principal de este tipo de redes es compartir información, sin importar a qué red estén conectados los dispositivos. [32]

- Existen distintas formas de montar servidores así como de crear la conexión entre cliente y servidor. Algunas de las herramientas/software que se encontraron que permiten montar un servidor, ya sea para recibir información o mandar información a un cliente son las siguientes:
 - Google Firebase: Es una plataforma que fue creada por Google que se enfoca en desarrollo, pudiendo servir tanto para creación de aplicaciones o para montar un servidor. Permite un mejor proceso de desarrollo, ya que reduce el tiempo de optimización y ayuda en la realización de pruebas y en la detección de errores. [57] Además puede almacenar todo esto en la nube, por lo que tanto para desarrollo de aplicaciones como para servidor, esto puede ser de utilidad.
 - XAMPP: Es un servidor web multiplataforma que ayuda también al desarrollo, creación y testeo de programas en un servidor local. Fue creado por Apache Friends y los usuarios pueden checar o alterar el código fuente conforme sea necesario. Este servidor constituye un entorno adecuado para poder revisar el funcionamiento de proyectos que estén basados en Apache, Perl, SQL, entre otros. Además, es compatible con todas las plataformas. [44]
 - Microsoft IIS: Es un servidor web flexible de Microsoft (Internet Information Services) que se corre en Windows para ejecutar las páginas o archivos HTML que se soliciten. Un servidor IIS puede aceptar solicitudes de dispositivos cliente remotos y devolver la respuesta adecuada. Esta cuestión permite que un servidor comparta y mande datos mediante redes de área local, así como intranets corporativas y redes de área amplia (WAN), via Internet. Un servidor web puede enviar datos a los usuarios de varias formas, como por ejemplo mediante páginas web HTML estáticas, mediante archivos, por ejemplo al cargarlos o descargarlos, así como documentos de texto, imágenes, entre otros. [49]
- En cuanto a la conexión cliente-servidor se encontró que existen módulos de MicroPython como ufirebase y también de Arduino IDE capaces de hacer requests a los servidores montados con las herramientas anteriormente mencionadas, más respecto sobre el montaje de los servidores y la conexión desde el cliente (auditor) en la sección de metodología.

Debido a que no se requiere comunicación entre auditores, sino más bien del tipo auditor-servidor, se optó por considerar que la red tipo cliente-servidor es la que mejor encaja a la situación problema, por lo que se buscará implementar esta para la solución de este reto.

2.6. Recursos hardware disponibles

Se buscará emular un entorno de comunicaciones, parecido al de la Figura 1 como se mencionó previamente. Teniendo en cuenta que se ocupará que el auditor funcione con recursos limitados y que se encargue de la recolección, encriptación y envío de los datos recolectados desde los dispositivos de medición de energía hacia el centro de control, se consideran los siguientes dispositivos apropiados para realizar el trabajo de auditor:

- Tarjetas Raspberry pi: Modelos a partir del Raspberry Pi 3 Model B (2016) que cuenten con antena WiFi.
- ESP32: Chip tipo SoC (System on a Chip), con WiFi integrado.
- Tarjetas de tipo Arduino con módulo WiFi.

2.7. Tabla Comparativa de Componentes

Cita	Dispositivo	Memoria	SRAM	Costo	Software Compatible
[19]	Arduino UNO R3	32 KB	2 KB	Original: 500 MXN, Genéricos: 300 MXN	Arduino IDE/Arduino CLI/Arduino Web Editor
[14], [16]	NodeMCU ESP8266	4 MB	64 KB	Menos de 200 MXN	C++ con ESPressif SDK o Arduino IDE, MicroPython con ESPressif Flash Download Tools
[27]	ESP32	448 KB ROM	520 KB	250 MXN	Arduino IDE/LUA/-MicroPython/Espres-sif IDF
[52]	Raspberry Pi Pi-co	2 MB	264 KB	200 MXN	C/C++ y MicroPython

Tabla 1: Tabla comparativa de especificaciones técnicas entre distintos dispositivos tipo SoC que pueden fungir como auditores.

En esta Tabla 1, se pueden observar en la primera columna los distintos dispositivos que se optó por utilizar para la comparativa. De igual manera se puede observar en la segunda columna las respectivas capacidades de memoria para cada uno de los mencionados dispositivos, así como sus capacidades de memoria SRAM en la tercera columna y el costo en pesos mexicanos para cada uno de estos dispositivos en la cuarta columna. Finalmente, la última columna de esta Tabla 1 brinda

información sobre la compatibilidad del software de los correspondientes dispositivos. Tras observar la información que brinda esta Tabla 1, se puede destacar que la ESP32 y la Raspberry Pi Pico presentan mejores capacidades de memoria y memoria SRAM que las demás, teniendo también un costo menor en comparación por ejemplo con Arduino UNO R3, por lo que estos dos dispositivos pueden ser buenos prospectos. Adicional a esto, se puede observar una columna inicial, en la cual se proporcionan las respectivas citas donde se puede obtener esta información para cada dispositivo, así como información adicional.

3. Propuesta

Para el caso de este reto, se optó primeramente por montar una arquitectura de red en la cual se tuviera un servidor con la información a encriptar en un archivo json. Una vez cargada la información esta será accedida desde un chip tipo SoC utilizando MicroPython, se optó por tener un servidor con la información a encriptar para así no tener que escribir las bases de datos sobre la tarjeta sino que los datos fueran accedidos en forma de peticiones conforme el programa los necesite. Una vez se tengan los datos específicos, se creará un token único de identificación para cada medición de la base de datos, estos tokens estarán estructurados como una cada de caracteres de la siguiente forma:

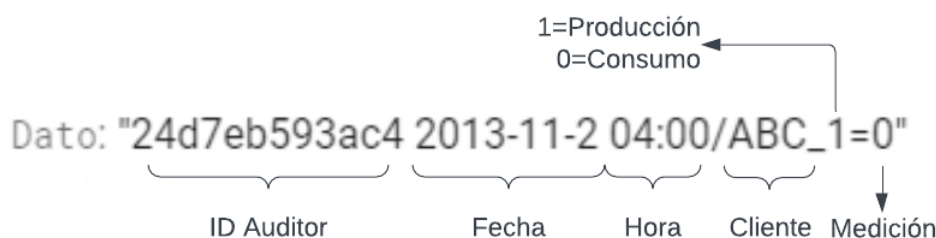


Figura 3: Estructura de los mensajes a ser enviados al centro de control desde el auditor. [15]

Una vez generada la cadena de datos se procederá a aplicarle los algoritmos de encriptación necesarios; para este caso se decidió que se va a utilizar tanto el SHA-256 como ECDSA. Por otra parte, en caso de utilizar un protocolo de comunicación https entre cliente-servidor, se tendrá que implementar algún tipo de algoritmo de cifrado sobre los datos, pues los datos pueden calificar como sensibles.

Una vez se hayan transformado los datos dentro del código se procederá a enviar estos en forma

de paquetes hacia un servidor destino, donde cada paquete contendrá la información del hasheo y autenticado generados por los algoritmos anteriormente mencionados. De igual manera, para facilitar el acceso a los datos ya en el servidor destino, se podrá acceder a cada paquete de datos por medio de una id pública, que simplemente será el índice que muestre si es un dato de consumo o producción, su índice de fecha y su índice de hora y estos parámetros serán los que se utilizarán para generar las id públicas.

Una vez se tengan los datos transformados almacenados en la segunda base de datos de firebase, un centro de control (que en nuestro caso será una computadora con los scripts correspondientes para poder acceder y monitorear los paquetes enviados), se encargará de contactar con la base de datos y de la misma forma, hacer request sobre esta para descargar todos los datos generados, uno de los datos almacenados en la base de datos va a ser la llave pública mediante la cual se podrá verificar si los mensajes son auténticos o no, sin necesidad de compartir ningún tipo de llave privada, aquí radica la característica de que ECDSA sea un algoritmo de cifrado por medio de clave pública, una vez que se tengan los datos descargados de la base de datos junto con la clave pública, el centro de control efectuara la parte de verificación de firmas del algoritmo ECDSA y mostrará si cada uno de los paquetes descargados son o no auténticos. En caso de que por alguna razón alguno de los paquetes descargados en el centro de control no sean verificados, se podrá enviar una notificación para alertar sobre esto.

El propósito principal de la materia del reto así como el reto en cuestión es el poder utilizar criptografía de clave pública para autenticar datos, por lo que el montaje de los servidores y la conexión entre estos y el cliente no busca seguir algún esquema específico, sino, que será libre de montar por parte del equipo. En la sección de metodología y desarrolla se mostrará a detalle la red construida para el intercambio de datos entre bases de datos, auditores y centro de control, claro que teniendo la idea de presentar al socio formador una solución completa y funcional se tuvo que asegurar que la red a implementar pudiera sin problemas mandar datos del auditor al centro de control.

Teniendo esto en cuenta no se pretende que el montaje de red tome un papel “secundario” en este documento, pero, debido a los temas impartidos en clase y teniendo en cuenta que era tan primordial el poder cifrar los datos como intercambiarlos entre dispositivos, en lo que este documento puede mostrar más énfasis es hacia temas relacionados directamente al algoritmo ECDSA, claro sin dejar de lado que se debe proponer una solución completa a la situación problema planteada.

La razón por la que se considera que el algoritmo ECDSA es muy buena opción para garantizar autenticidad entre cliente y servidor es que este se basa en el problema matemático de logaritmo elíptico, también llamado problema del logaritmo discreto en curvas elípticas. Justamente este problema ayuda a que este método aplicado a la criptografía se considere bastante seguro, ya que es muy complicado calcular el logaritmo discreto de un elemento de curva elíptica aleatoria respecto a un punto base que se conoce de manera pública, por lo que se pueden usar las funciones de este método para encriptar o generar firmas de autenticado entre pares, una vez hecho esto, se vuelve muy difícil

de conocer los datos generadores del output del algoritmo, por lo que la capacidad para un tercero malicioso de poder descryptar se vuelve muy limitada [50]. Ya que se tomó la decisión de optar por utilizar esto, ahora se debe de tomar en cuenta que se deberá considerar el aspecto de la creación de la clave por medio de precisamente el uso de las curvas elípticas. En este caso, considerando a un remitente que pretenda mandar un mensaje a su destinatario, deberá junto con este establecer los parámetros para la curva elíptica. Para esto se buscaría que las partes o elementos involucrados en la comunicación establezcan el campo, ecuación de curva y punto G . Tras esto, el remitente genera una llave pública y una privada. Asimismo, en el aspecto de firma digital también se puede aplicar esto, pues el algoritmo de firma digital de curva elíptica, o ECDSA por sus siglas en inglés, es un algoritmo de firma digital que hace uso de curvas elípticas y es muy comúnmente empleado para criptografía de clave pública [59].

A continuación se proporciona un pequeño resumen del funcionamiento de cada uno de los algoritmos a implementar:

3.1. ECC (Elliptic-Curve Cryptography)

Los algoritmos que utilizan curvas elípticas son reconocidos por poder generar clavez robustas y ser rápidos de computar, a comparación de otros algoritmos. Estos pueden utilizar tanto clave pública como privada, y además de la protección que puede brindar este algoritmo, es también práctico y eficiente como se mencionó anteriormente. Es comúnmente utilizado en tecnología blockchain ya que puede brindar una alta protección y cumplir con los niveles de velocidad requeridos para que este sistema sea fluido. Consiste en un sistema de cifrado asimétrico, utilizando una estructura de álgebra de curvas elípticas sobre campos finitos, permitiendo que las claves puedan ser más concisas, pero aún así bastante seguras. Se basa en el objetivo de que se haga un proceso de operaciones algebraicas que no sean tan complicadas de ejecutar para el encriptado, pero que sean bastante complicadas de poder revertir, pues entre más difícil sea esto, más protegidos estarán los datos que se encriptaron, volviéndose más difícil y tardado el proceso de descryptado para los posibles ciberatacantes. [39]

3.2. ECDSA (Elliptic Curve Digital Signature Algorithm)

El algoritmo de firma digital de curva elíptica (ECDSA) es un algoritmo de firma digital (DSA) que utiliza claves derivadas de la criptografía de curva elíptica (ECC). Es una ecuación particularmente eficiente basada en criptografía de clave pública (PKC).

ECDSA se usa en muchos sistemas de seguridad, es popular para su uso en aplicaciones de mensajería segura y es la base de la seguridad de Bitcoin (con "direcciones" de Bitcoin que sirven como claves públicas).

ECDSA también se usa para Transport Layer Security (TLS), el sucesor de Secure Sockets Layer (SSL), mediante el cifrado de conexiones entre navegadores web y una aplicación web. La co-

nexión cifrada de un sitio web HTTPS, ilustrada por una imagen de un candado físico que se muestra en el navegador, se realiza a través de certificados firmados mediante ECDSA. [26]

3.3. SHA-256

Para poder implementar ECDSA de forma completa, se requiere de algún tipo de hasheo que transforme el mensaje con los datos originales a alguna representación numérica, se optó por utilizar SHA-256 debido a la amplia documentación respecto a este algoritmo, también se tomó en cuenta que no es considerado un algoritmo lento de computar, claro, puede ser más tardado que otros algoritmos de hashing como el SHA-1 o MD5, sin embargo, la diferencia de tiempo puede llegar a ser muy pequeña, sin llegar a afectar el funcionamiento del programa en la mayoría de casos, este algoritmo retorna una cadena de 256 bits con el objetivo de verificar la integridad del documento, texto, archivo, etc. pues con el más mínimo cambio en el contenido de este, la cadena que se genera con SHA-256 debería ser totalmente diferente, sin importar el grado en el que se haya cambiado la información de los datos originales.

Detalles sobre la implementación de ambos algoritmos se pueden consultar en la sección de metodología y resultados.

Para esta situación problema, se decidió que las tarjetas del tipo ESP32 fungirán como auditores para la situación problema debido a sus componentes y precio, por otra parte la implementación de micro-chips para este tipo de monitoreos parece ser ideal, ya que a comparación de otros dispositivos como por ejemplo una Raspberry Pi 4B, estos pueden ser más fáciles de implementar y mucho más baratos y fáciles de conseguir. Por otra parte, el uso de lenguajes como C/C++ y MicroPython pueden facilitar al equipo la implementación de los algoritmos criptográficos, claro, se tendrán limitaciones ya que MicroPython no cuenta con la misma cantidad de módulos y librerías disponibles que en Python, sin embargo, se tienen en cuenta estas limitaciones y se planea trabajar sobre estas en el desarrollo de la solución propuesta en el documento.

A continuación se muestran las versiones de los lenguajes de programación disponibles a utilizar en la ESP32:

Cita	Lenguaje	Versión
[27]	C++	CMake (at least version 3.13) cmake_minimum_required(VERSION 3.13)
[27]	Python	micropython-20220618-v1.19.1

Tabla 2: Lenguajes disponibles para programar dentro de la ESP32, así como sus respectivas versiones.

En la primera columna de esta Tabla 2 se encuentran las citas de donde se obtuvo la informa-

ción que se proporciona en esta tabla. La segunda columna muestra los lenguajes disponibles para la programación de la tarjeta ESP32, por el momento se opta por usar tanto el lenguaje de C++ como Python. Finalmente, en la última columna, se especifica la respectiva versión de cada uno de los lenguajes de programación.

Debido a las limitaciones de recursos que presenta los chips tipo SoC, podemos ver que las versiones de los lenguajes para programar dentro de la ESP32 son versiones limitadas en cuanto a librerías y funciones disponibles de los lenguajes.

También se decidió utilizar la herramienta Firebase fungiendo como servidor de base de datos de los datos originales y como servidor de los datos ya transformados, varias de las ventajas que ofrece Firebase es que existen distintas librerías en MicroPython para poder conectar y manipular al servidor, también cabe mencionar que la información almacenada en el servidor se actualiza en tiempo real y que cuenta con versiones de prueba, lo cual nos puede facilitar la implementación de prototipos de servidores.

Los protocolos de seguridad específicos a usar en la arquitectura de comunicación entre auditor y centro de control se verán a detalle en partes posteriores del documento. En esta sección mayormente se muestra la arquitectura y el flujo de información que se busca que tenga la red a implementarse.

Se buscará que el esquema criptográfico a implementar cuente con las cualidades de encriptar la información, firmarla y verificarla digitalmente, a continuación se muestra un diagrama que busca ilustrar este proceso mediante el uso de los lenguajes de programación disponibles a utilizar para esta situación.

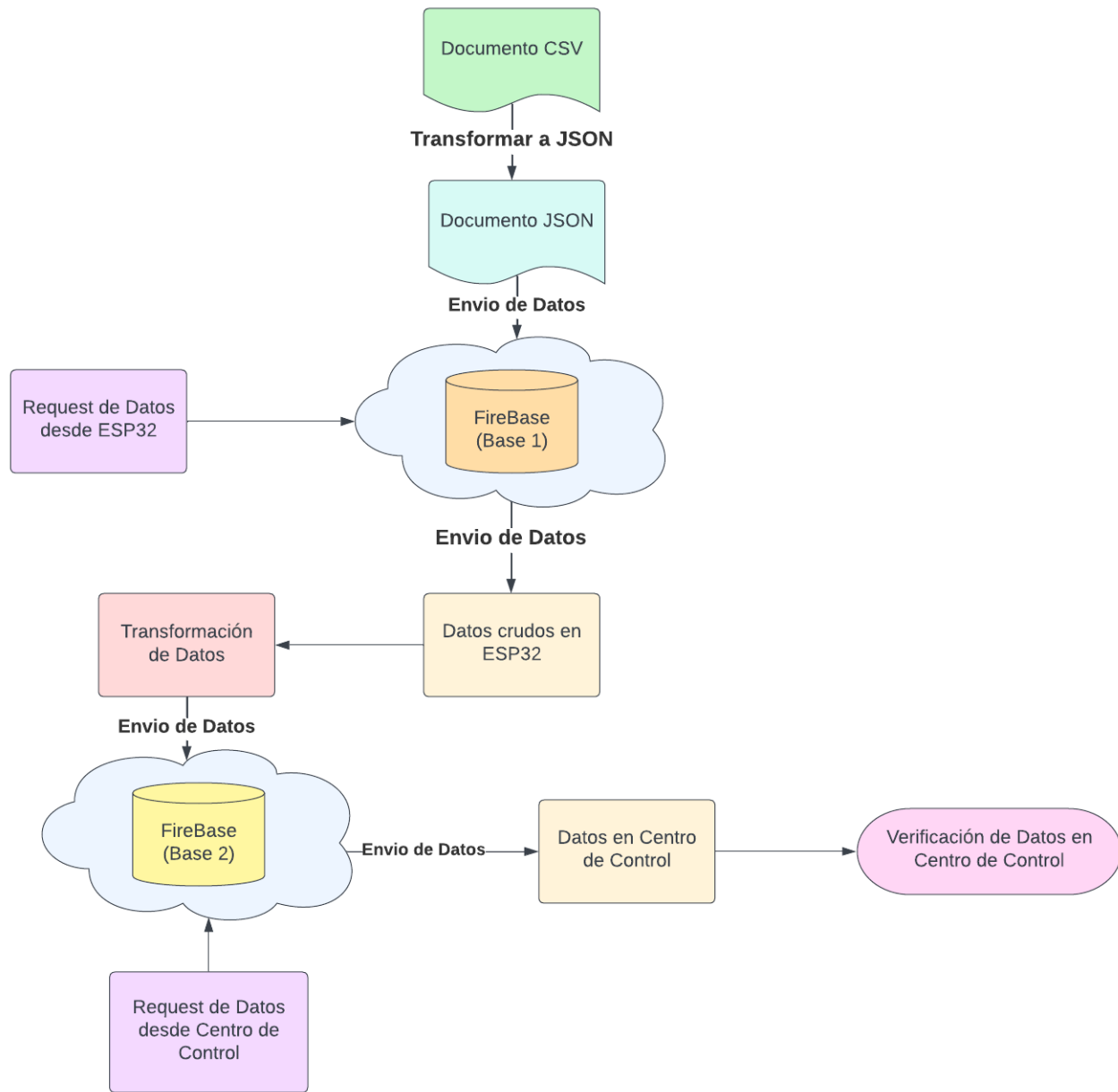


Figura 4: Esquema de simulación de la ESP32.

En esta Figura 4, se puede observar el esquema de simulación de la ESP32 que se va a utilizar para esta situación problema, siendo en este caso la ESP32 el dispositivo que se encuentra justamente en el centro del esquema como se puede observar, el cual funcionará como auditor. Con respecto a esta ESP32, se llevará a cabo firmado digital y validación, para lo cual será necesario un script de

Python (más específicamente MicroPython). Se puede observar el proceso de flujo de los datos desde la su forma original representando trazas en un archivo csv, hasta el almacenamiento final de sus firmas y hashing correspondientes por dato.

El lenguaje de programación Python permitirá que se elabore el script de firmado así como de validación, como se mencionó previamente, en nuestro caso el centro de control será el servidor que reciba los datos debido a que se va a buscar que por medio de sus recursos propios, pueda validar las firmas recibidas por la ESP32 y también pueda tener conexión directa con la ESP32 y manipular los datos que se le almacenan.

Debido a que se pueden dividir los datos de la base de datos en tipo Producción y Consumo, optamos por simular que se tuvieran dos auditores, donde cada uno de los auditores midiera solo un tipo de dato, y se mandaron estos al mismo servidor como se muestra en la siguiente imagen.

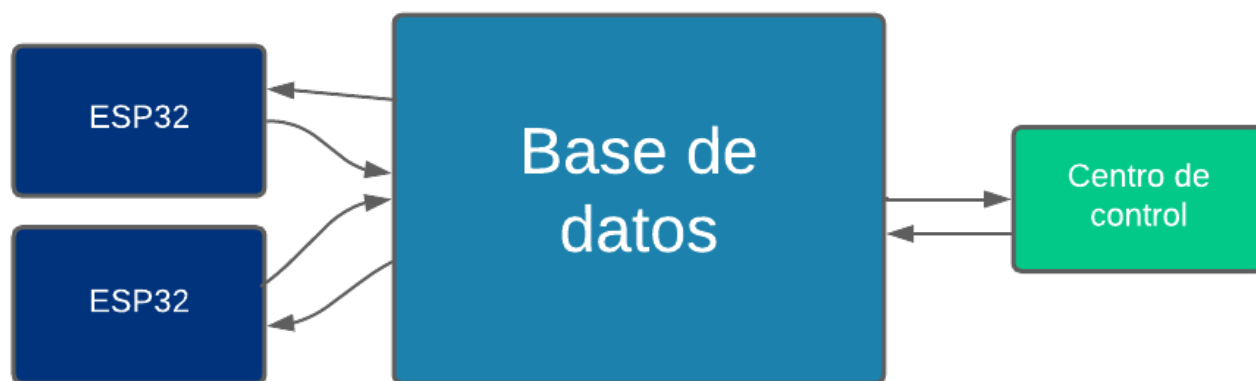


Figura 5: Esquema de comunicación entre auditores y centro control

En esta Figura 5, se puede observar cómo el servidor que recibe los datos de ambos auditores es capaz también de comunicarse con ellos en caso de ser necesario. Cabe recordar que cada tarjeta ESP32 se va a encargar de manejar un solo tipo de dato para el esquema.

Antes de comenzar a programar, se realizó una búsqueda de las librerías que podrían ser de ayuda. Estas librerías se pueden ver en la siguiente tabla:

Lenguaje	Librería	Versión
Python	random	Depende de versión de Python.
Python	binascii	Depende de versión de Python.
Python	cryptography	Depende de versión de Python.
Python	hashlib	Depende de versión de Python.
Python	datetime	Depende de versión de Python.
Python	json	2.0.3
Python	time	Depende de versión de Python.
Python	toml	Depende de versión de Python.
Python	os	Depende de versión de Python.
Python	pandas	1.3.5
Python	shutil	Depende de versión de Python.
Python	subprocess	Depende de versión de Python.
Python	typer	0.7.0
Python	serial	Depende de versión de Python.
Python	pytest	3.6.4
Python	from __future__ import annotations	future-0.18.3 iso8601-1.1.0 serial-0.0.97.
Python	from pathlib import Path	1.0.1
Python	from io import BytesIO	3.3
Python	from types import TracebackType	1.0b10
Python	from base64 import b64decode	1.2.3
Python	from base64 import b64encode	1.2.3
Python	from typing import Any	0.0.1
Python	from hashlib import sha256	1.4.0
Python	from time import sleep	3.6
Python	from pico_crypto_key import CryptoKey	1.4.0
Python	from pico_crypto_key import b64_to_hex_str	1.4.0
Python	from pico_crypto_key import hex_str_to_b64	1.4.0
C++	thingProperties	1.0.4
C++	Arduino_ConnectionHandler	0.7.3
C++	DHT sensor library for ESPx	1.18.0
C++	ArduinoIoTCloud	1.0.0

Tabla 3: Esta tabla incluye las librerías que se pueden utilizar dentro de la solución propuesta. Se especifica cuáles se utilizaron en la sección 4 del documento.

Esta Tabla 3, en su primera columna brinda información sobre los lenguajes de programación a utilizar, para posteriormente especificar las librerías que se utilizarán de dicho lenguaje, siendo en este caso la mayoría de las librerías específicamente del lenguaje Python como lo indica la primera

columna de la tabla con su título, aunque también se incluyen algunas librerías del lenguaje C++. Posteriormente, la segunda columna de esta Tabla 3 se encarga de nombrar y dar a conocer cada una de las librerías que se utilizarán de dichos lenguajes. Finalmente, la última columna de la tabla especifica la versión de cada una de las mencionadas librerías a utilizar.

4. Metodología y Desarrollo

El esquema que se seguirá como guía para el desarrollo de la propuesta se muestra a continuación. Tener este esquema permite de mejor forma saber qué se necesita hacer, cómo y cuándo, tal como se puede observar a continuación:

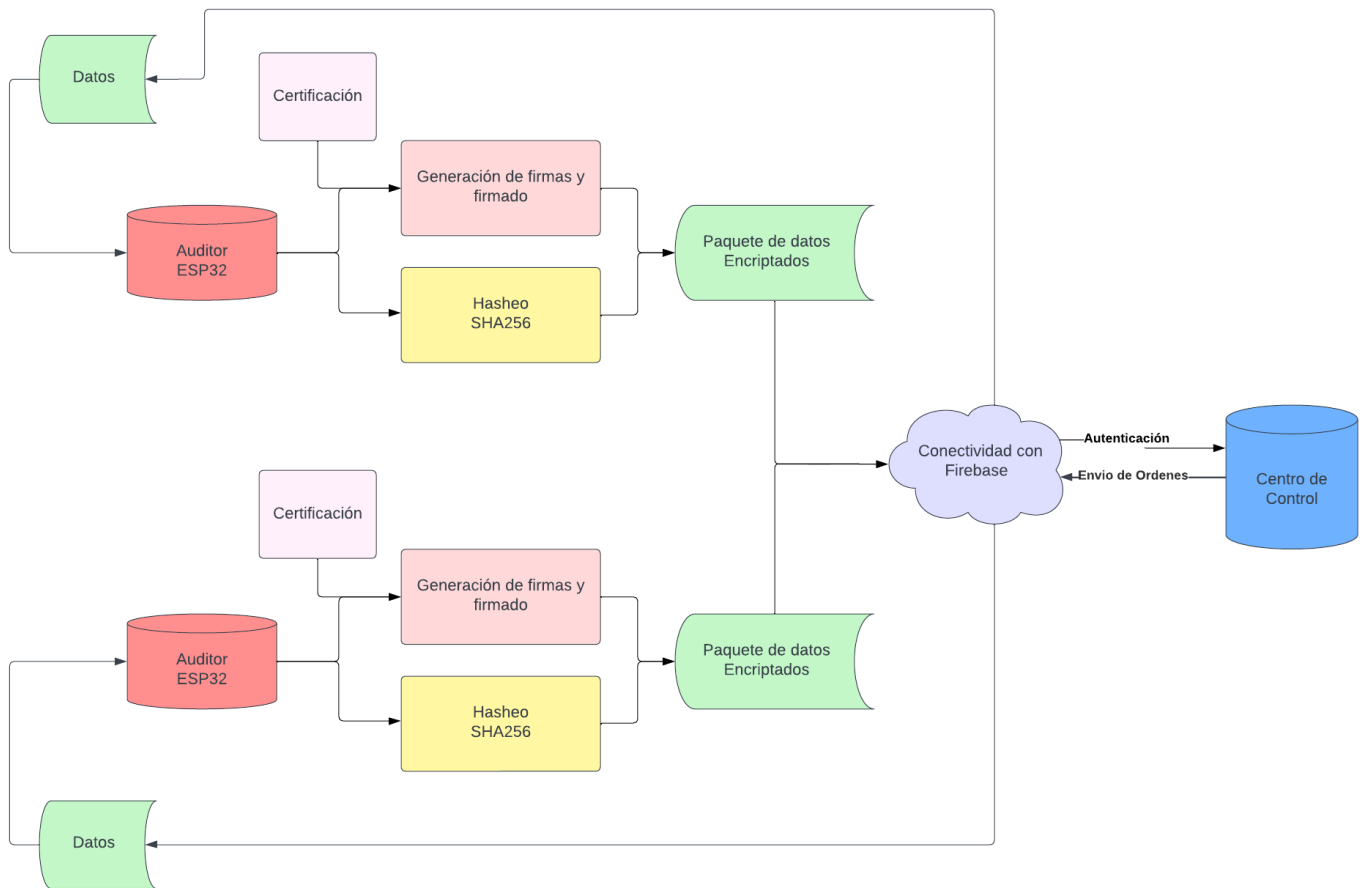


Figura 6: Diagrama de implementación.

En esta Figura 6, se puede observar el diagrama de implementación, el cual es una representación del flujo de datos con sus respectivas transformaciones. El auditor (rojo) mandará la información en cada cierto tiempo, los datos que este recibe para la funcionalidad y disponibilidad de la solución del reto, son mandados desde el centro de control para simular que el auditor recibe de un aparato externo los datos, después estos datos pasarán por una creación de certificado (rosa), una generación de firmas y firmado digital (naranja) y un hasheo (amarillo) para poder tener un mensaje seguro (verde). Después se hace la conexión a la red y a firebase para poder desplegar los datos en el centro de control (azul). Asimismo, cabe mencionar que aunque en esta Figura 6, hay una conexión intermediaria a manera ilustrativa entre centro de control (azul), con datos (verde) y auditor (rojo), cabe aclarar que no forma parte de la intención del diagrama el indicar que los datos se intercambien específicamente de esa manera. Ahora bien, ya teniendo lo anterior en mente, a continuación se mencionará más a detalle lo que se hizo para realizar el esquema.

Para llevar a cabo este esquema, se levantó una conexión del tipo cliente-servidor donde se pudieran enviar datos de la placa ESP32 a Firebase y por otra parte se implementarán 2 algoritmos criptográficos dentro de la placa ESP-32 con el propósito de cifrar los paquetes de datos.

A continuación, se muestran los pasos que se siguieron para montar la arquitectura de red mediante la cual se dará el flujo de datos desde la base de datos hasta el centro de control final.

4.1. Transformación de Datos a json

Primeramente, se decidió clasificar los datos de la base de datos en base a la columna de “Consumo (0)/Producción(1)”, por medio de la herramienta sort en Excel, seguido de esto se decidió crear una hoja de cálculo para las trazas de consumo y otra para las trazas de producción, con el propósito de que cada base de datos nueva fuera procesada por su respectivo auditor.

Debido a que la base de datos de Firebase en tiempo real maneja datos en formato json, se tuvo que, de igual forma transformar los datos en formato csv a json, para esto se utilizó la librería de Pandas en Python, el código fuente para la transformación de datos, así como los datos transformados puede ser consultado aquí [2], el acceso será restringido por lo que aquellos que quieran acceder a la carpeta tendrán que solicitar permiso al propietario.

La transformación se realizó de tal forma que cada elemento dentro del archivo json, contuviera la información de una de las filas del archivo csv, debido a que solo se puede importar un archivo json a firebase, ya que de caso contrario la información se sobrescribe, se decidió juntar ambos archivos json con la función `pandas.DataFrame.append()`, se le agregó un prefijo a cada dato del archivo csv para que se pudiera diferenciar si este era del tipo de producción o consumo. Por lo que al final, se obtuvo un archivo json de 728 elementos, cada uno conteniendo 101 datos, correspondientes a las mediciones o de consumo o de producción a través del día.

4.2. Montaje de Base de Datos y Almacenamiento de Datos

Se accedió a la herramienta de firebase desde el siguiente link:

<https://console.firebase.google.com/>, seguido de esto se creó un proyecto y se accedió a la herramienta Realtime Database. Después, se creó la base de datos en test mode, para así poder fácilmente acceder a los datos para los primeros prototipos de arquitectura de red. Ya una vez creada la base de datos, se importaron los archivos json generados a la base de datos, tal como se muestra en la Figura 7:

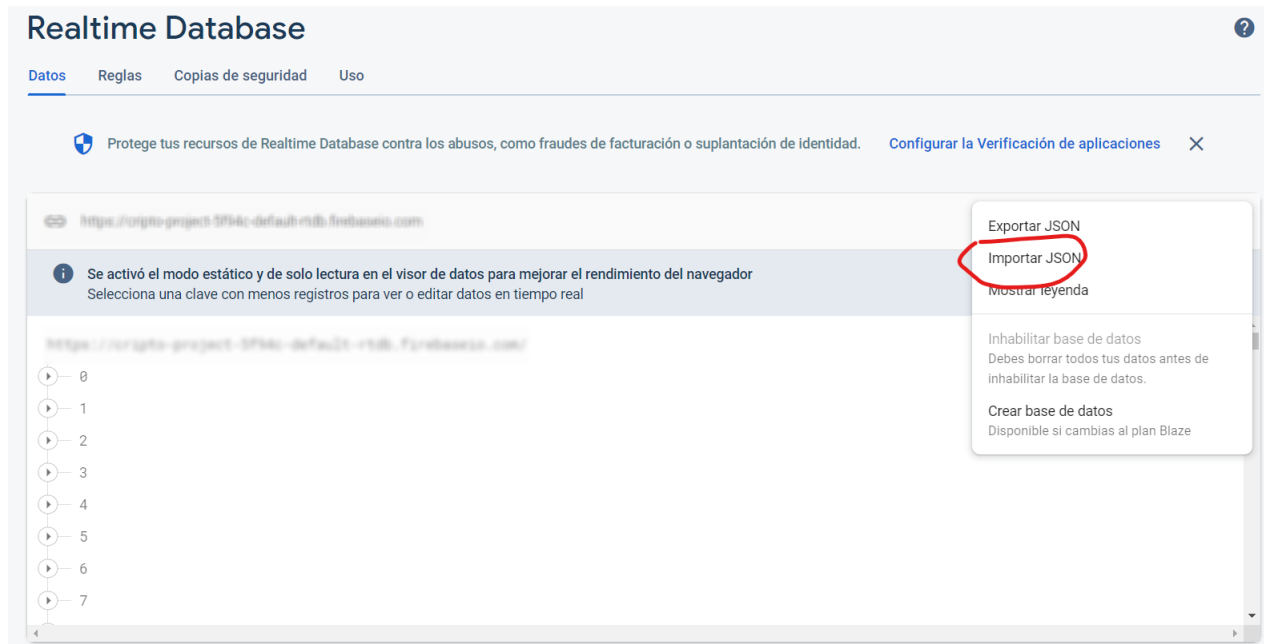


Figura 7: Opción para importar bases de datos a firebase.

Debido a que esta base de datos ya había sido previamente cargada con sus respectivos datos, aparecen los índices de los datos en la imagen, sin embargo, si es la primera vez que se cargan los datos en la herramienta, la consola aparecería vacía.

Cabe mencionar que los datos únicamente pueden ser accedidos desde la cuenta que creó la base de datos. Una vez importadas ambas bases de datos a firebase se pudo proceder a montar el centro de control.

4.3. Montaje de Segunda Base de Datos

El montaje de esta base es muy similar al paso anterior debido a que se utiliza de igual forma firebase para almacenar los paquetes de datos transformados, la mayor diferencia entre ambas bases

de datos es que a la base de datos que contiene los datos originales, se le hacen requests de datos, mientras que en el centro de control se escriben los datos transformados.

Para mostrar de mejor forma que la conexión entre auditores y bases de datos pueden ser totalmente remotas independientemente de la red a la que se este conectado alguno de los componentes, la segunda base se creó en otra cuenta diferente a la cuenta con la que se creó la base de los datos originales, sin embargo un usuario de una cuenta no puede modificar la base de datos creada con la otra cuenta y viceversa.

La segunda base de datos contará con un total de 728 datos, todos en formato json, donde cada dato contendrá la cadena de datos a asegurar, así como su hash y datos de firmado, debido a que el proyecto busca ilustrar el funcionamiento correcto de los algoritmos de hash y autenticación, en la base de datos de centro de control se incluye la string de datos original por si se quisiese comprobar o hacer cambios en los algoritmos de ciberseguridad en un futuro.

4.3.1. Implementación de MicroPython en ESP32

Se procedió a instalar MicroPython para ESP32 desde la página oficial de MicroPython, la cual es <https://micropython.org/download/esp32/>. Esto para instalar los módulos de MicroPython y cargar código a la ESP32 desde Mu Editor, se siguió un tutorial disponible en YouTube en el canal iobotic, que puede ser consultado aquí [34].

Una vez teniendo la conexión entre la ESP32 y Arduino Cloud, y también teniendo instalados los módulos de MicroPython dentro de la tarjeta, se puede proceder a implementar el código necesario para leer los datos e implementar los algoritmos criptográficos por los que se optaron. Detalles sobre esto en las siguientes secciones.

4.4. Costos Firebase

Para calcular los costos se deben de ver los planes que Firebase maneja. Cabe mencionar que Firebase utiliza la herramienta Firestore para el almacenamiento. Cuando se hace uso de esta herramienta se cobra por lo siguiente[20]:

- El número de documentos que lees, escribes y eliminas.
- El espacio de almacenamiento utilizado por tu base de datos, incluyendo los costos de funcionamiento de metadatos e índices.
- El espacio de almacenamiento utilizado por tu base de datos, incluyendo los costos de funcionamiento de metadatos e índices.

Cabe mencionar que el uso de este ancho de banda y almacenamiento se calcula en gigabytes (con la equivalencia de $1 \text{ GB} = 2^{30} \text{ bytes}$). El cargo se aplica cada 24 horas. En la siguiente tabla se pueden visualizar los precios por operaciones de escritura, eliminación, almacenamiento y lectura para cada tipo de ubicación dentro de Firestore[20].

		Cuota gratuita por día	Precio una vez superada la cuota gratuita (por unidad)	Unidad y precio
[20]	Operaciones de lectura de documentos	50,000	\$0.06	por 100,000 documentos
[20]	Operaciones de escritura de documentos	20,000	\$0.18	por 100,000 documentos
[20]	Operaciones de eliminación de documentos	20,000	\$0.02	por 100,000 documentos
[20]	Datos almacenados	1 GB de almacenamiento	\$0.18	GB/mes

Tabla 4: Precios firestore.

Los precios de la tabla anterior que fue obtenida de la documentación oficial de Firestore vienen dados en dólares americanos. De igual manera, si pagas con una moneda distinta a esta, se aplicarán los precios correspondientes a dicha moneda [53].

Ahora bien, retomando lo anterior, si el consumo no sobrepasa el límite establecido en las cuotas gratuitas Firebase detectara que estás en el plan Spark, el cual es uno de sus dos planes. Cabe mencionar que solo existen dos tipos de planes, el Spark y el Blaze [18]:

■ Spark:

- Plan al que te mete Firebase al utilizar sus servicios y en el cual estarás mientras no sobrepases las cuotas de lectura, escritura, eliminación y almacenamiento diario, mostrados en la Tabla 4.
- Uso completo de los productos y funciones gratuitos de Firebase (como métodos de inicio de sesión social, FCM y Crashlytics).

- Cuotas de uso sin costo para productos de pago de Firebase (como Cloud Firestore, Cloud Storage y Hosting).
- **Blaze:**
 - Todo lo anterior mencionado en el plan Spark.
 - Precios de pago por uso para cualquier uso adicional de productos de pago de Firebase.
 - Cuota de uso sin costo para Cloud Functions para Firebase, luego precios de pago por uso.
 - Acceso a productos y funciones de pago de Google Cloud (como Pub/Sub, Cloud Run o transmisión de BigQuery para Analytics).

Una vez aclarado lo anterior, lo más recomendable es calcular qué tantos recursos se consumirán. Posterior a esto, si vemos que pasamos la cuota gratuita por día mostrada en la Tabla 4 debido al tamaño de la base de datos manejada en este reto, entonces no entramos en el plan Spark, por ende lo que queda es entrar en el plan Blaze.

En el siguiente link <https://firebase.google.com/pricing?hl=es-419#blaze-calculator> se dispone de una calculadora oficial de Firebase, con la cual se puede obtener un aproximado de los gastos que se podrían llegar a generar.

4.4.1. Aproximado de costos para la situación problema

Teniendo en cuenta que el tamaño de la base de datos 1 es de 675 KB, y que para la segunda base de datos cada paquete pesa en promedio $\frac{159}{572,900} MB$ en donde el numerador(159) hace referencia al peso de la base de datos y el denominador(572,900) hace referencia al numero de datos dentro de esta, esto para calcular cuanto pesaba cada datos en promedio. Ahora sí, retomando lo anterior, se tiene un total de 69888 paquetes, debido a que se tienen 728 días de información donde cada día cuenta con 96 registros. El peso total de la segunda base de datos sería de 19.39 MB, por lo que, para ambos casos no se pasaría de la cuota gratuita de almacenamiento de 1 GB por día. Por otro lado, se puede ver en la página de costos [23] que se tiene un límite gratuito de 360 MB de datos a transferir por día por proyecto. Para el caso de este proyecto, la cantidad total de datos transferidos a lo largo de todos los días en los que se trabajó con la base de datos 1 fue de 2.3 MB y de 56.45 MB para la base de datos 2.

Se puede ver de esta forma que la cantidad de datos almacenados y transferidos para la red montada, teniendo "dos auditores"(en este caso se tuvo un solo ESP32 como auditor, sin embargo, este fungió por el momento como auditor de datos de consumo y de producción ya que tenía ambos scripts de monitoreo implementados) no está cerca de pasar el límite gratuito ni en almacenamiento

ni en transferencia de datos. Como prueba de esto, se muestra la gráfica de carga total de datos transferidos que tiene la segunda base de datos (la base de datos más pesada), en la Figura 8:

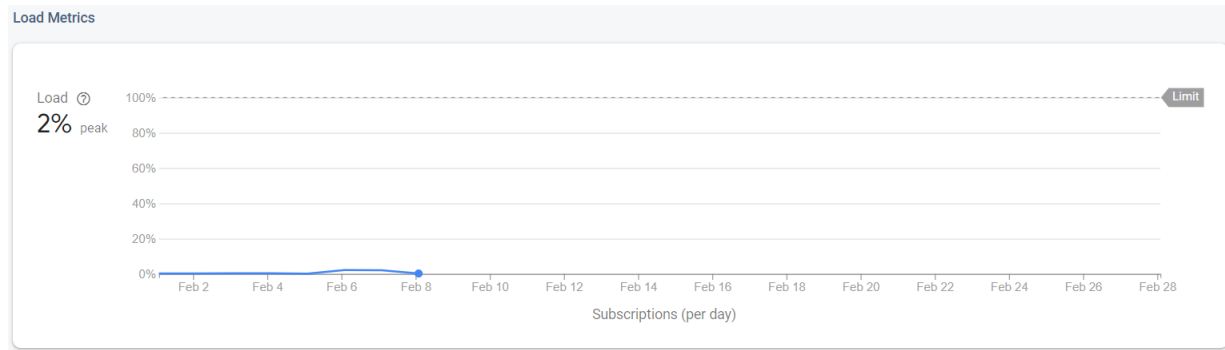


Figura 8: Total de datos transferidos en la segunda base de datos.

Por propósitos de escalabilidad, suponiendo que se tienen 100 auditores y que cada auditor manda por día 96 paquetes de datos, donde cada uno de estos paquetes de datos en promedio tiene un peso de $\frac{159}{572,900}$ MB, la cantidad de datos a almacenar por día sería de 2.66 MB en promedio. De igual manera, estos deberán ser recibidos por la base de datos y transferidos hacia el centro de control, por lo que en promedio la cantidad de datos en transferencia al día sería de 5.32 MB. En caso de que los datos sean almacenados también desde el centro de control, digamos en disco duro, los datos en la nube del día podrían ser eliminados (no sería lo más recomendable, sin embargo, disminuiría la cantidad de datos almacenados en la nube), debido a la naturaleza del tipo de datos almacenados que son cadenas de caracteres, podemos ver en primera instancia que Firebase maneja bien la cantidad de datos sin sobrepasar el límite gratuito. Claro que en algún punto, si se sigue aumentando la cantidad de información a almacenar y transferir, se pasará de la cuota gratuita, sin embargo se cree que para menos de 100 auditores no debería de ser problema el almacenamiento y la transferencia de datos a través de Firebase.

4.5. Conexión de ESP32 a Base de Datos y Centro de Control

La conexión entre la ESP32 y ambas bases de datos se realizó utilizando MicroPython. Prácticamente, el programa se encarga de realizar requests para obtener datos de un servidor y realizar request para escribir datos sobre otro servidor, se podría decir que el programa para el manejo de trazas de consumo es idéntico al de manejo de trazas de producción, sin embargo, lo único que cambia es, los datos a los cuales se accede por medio de cambiar, simplemente cambiando un poco el index del bucle for que hace el request de datos, se puede acceder ya sea a los datos de consumo, o a los datos de producción.

De igual manera, se puede acceder a ambos scripts desde aquí [2]. El script que maneja los

datos de trazas de consumo se llama `main_Consumo.py` y el que maneja las trazas de producción se llama `main_Produccion.py`.

Ciertas funciones dentro de estos dos scripts, en específico las funciones `get_wifi_ssid()` y `get_wifi_password()` no están definidas para que retornen valores reales dentro del repositorio de github [2], por lo que en caso de querer correr el código de estos dos scripts se deberá de cambiar a los respectivos datos de la red a la que se quiera conectar el ESP32.

4.6. Esquema Criptográfico, implementación de SHA-256 y ECDSA

4.6.1. Librerías y recursos a utilizar

Se utilizaron únicamente librerías de MicroPython para el auditor y librerías de Python en el centro de control. Las librerías que se utilizaron fueron las siguientes.

Para los auditores (ESP32):

- `ufirebase`: Librería para hacer requests de datos a firebase desde Micro Python. Esta librería fue utilizada para hacer get y put request en ambas bases de datos montadas.[7]
- `os`: Permite interactuar directamente con el sistema operativo que esta siendo usado. Esta se uso para conectar el micro-controlador a una red wifi.[10]
- `network`: Este módulo proporciona controladores de red y configuración de enrutamiento. También se utilizó con el propósito de conectarse a la red WiFi.
- `time`: Proporciona funciones relacionadas con tiempo. Con esta se podía saber de mejor manera si la conexión a internet se había realizado de forma exitosa.
- `hashlib`: Librería con algoritmos de hashing. Se utilizó para implementar SHA-256.[8]
- `ubinascii` y `binascii`: Librerías para convertir datos binarios a datos ASCII y viseversa. Fue un auxiliar para convertir las strings de hashes generados de hexadecimal a decimal.[9]
- `ECDSA_signature` (librería propia): Esta librería fue implementada por cuenta propia, esta librería tiene algoritmos para realizar operaciones de curvas elípticas como suma de puntos y multiplicación de un escalar por un punto, la librería puede ser consultada en el repositorio creado para este proyecto aquí. [2]

Para el centro de control:

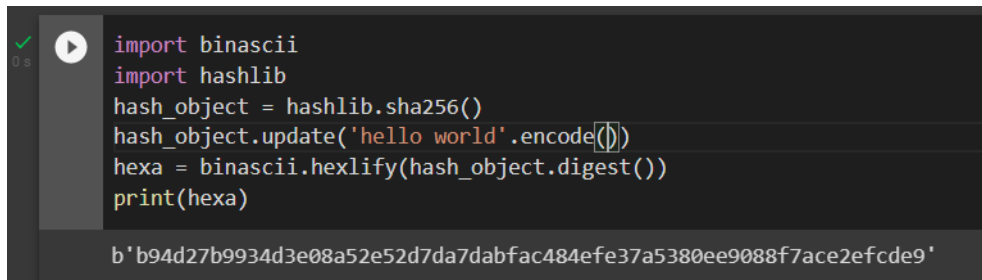
- `Hashlib`: Librería con algoritmos de hashing. Se utilizó para implementar SHA-256 debido a que es necesario tener el algoritmo de hasheo en la verificación de firmas.[8]

- binascii: De igual forma se tuvo que hacer un procesamiento de datos de hexadecimal a decimal, esta librería tiene funciones para eso.[9]
- firebase_admin: Librería para verificar credenciales de autenticidad para acceder a bases de datos de firebase.[7]
- ECDSA_signature (librería propia): Esta librería fue implementada por cuenta propia, esta librería tiene algoritmos para realizar operaciones de curvas elípticas como suma de puntos y multiplicación de un escalar por un punto, la librería puede ser consultada en el repositorio creado para este proyecto aquí. [2]

4.6.2. SHA-256

Debido a que no pudimos ver como funcionan los algoritmos de hashing como el SHA-256 en las clases para realizar este reto, se tuvo autorización para usar recursos externos como librerías para implementar el algoritmo de hashing dentro de los códigos de procesamiento de trazas dentro de los auditores. Para nuestro caso utilizamos la librería hashlib y binascii, ambas implementadas por default ya dentro de MicroPython. En cualquiera de los dos códigos proporcionados para el manejo de trazas, se puede encontrar de forma comentada qué secciones de código se encargan de hashear la información, y también se hace la referencia en caso necesario para indicar dónde se encontró la información base para el código.

En ambos códigos se crea un objeto del tipo hashlib.sha256 con la capacidad de que se le alimente una cadena de caracteres y devuelva la cadena de hasheo en formato hexadecimal. Un ejemplo de esto puede ser Figura 9.



```

import binascii
import hashlib
hash_object = hashlib.sha256()
hash_object.update('hello world'.encode())
hexa = binascii.hexlify(hash_object.digest())
print(hexa)

b'b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9'

```

Figura 9: Proceso que sigue el auditor para hashear datos, la string proporcionada es para mostrar un ejemplo del funcionamiento.

4.6.3. ECDSA

Para el aspecto de la generación de clave con curvas elípticas, en el caso de que un remitente desee enviar un mensaje o documento a un destinatario, deben ponerse de acuerdo sobre los parámetros de la curva. Además del campo y la ecuación de la curva, se necesita G , un punto base de primer

orden en la curva; y n es el orden multiplicativo del punto G , donde:

- $CURVA$: El campo de la curva elíptica y la ecuación utilizada.
- G : Punto base de la curva elíptica, un punto en la curva que genera un subgrupo de gran orden primo n .
- n : Orden entero de G , lo que significa que $n \times G = O$, donde O es el elemento identidad.
- d_A : Llave privada (Seleccionada aleatoriamente).
- Q_A : Llave pública $d_A \times G$ (Calculada mediante curvas elípticas).
- m : El mensaje o documento.

El orden n de la base del punto G debe ser primo. Se asume que cada elemento diferente de cero del anillo $\mathbb{Z}/n\mathbb{Z}$ es invertible, por lo que $\mathbb{Z}/n\mathbb{Z}$ debe ser un campo. Esto implica que n debe ser primo. [40]

Posteriormente, el remitente crea un par de claves, que constan de un entero de clave privada d_A seleccionadas aleatoriamente del intervalo $[1, n - 1]$; y una clave pública $Q_A = d_A \times G$. Se usa \times para denotar la multiplicación de puntos de curva elíptica por un escalar. Para que el remitente firme un mensaje m se deben realizar las siguientes operaciones:

1. Calcular $e = HASH(m)$. Donde $HASH$ es una función hash criptográfica, como SHA-2, con la salida convertida a un número entero.
2. Dejar z ser el L_n bits más a la izquierda de e , donde L_n es la longitud en bits del orden de grupo n . Nótese que z puede ser mayor que n , pero no puede tener una mayor longitud.
3. Seleccionar un entero aleatorio criptográficamente seguro k de $[1, 1 - n]$
4. Calcular $r = x_1 \text{ mód } n$. Si $r = 0$ es necesario regresar al tercer paso.
5. Calcular $s = k^{-1}(z + rd_A) \text{ mód } n$. Si $s = 0$ es necesario regresar al tercer paso.
6. La firma es el par (r, s) . Y $(r, -s \text{ mód } n)$ también es una firma válida.

Como señala la norma, no sólo se requiere para k ser secreto, pero también es crucial seleccionar diferentes k para diferentes firmas, de lo contrario, la ecuación en el paso 6 se puede resolver para d_A , la clave privada, dadas las dos firmas (r, s) y (r, s^t) , empleando la misma incógnita k para diferentes mensajes conocidos m y m^t , un atacante puede calcular z y z^t , y desde $s - st = k^{-1}(z - z^t)$. Es importante destacar que las operaciones anteriormente mencionadas se realizan en módulo n ; y el

atacante puede encontrar $\frac{z-z^t}{s-s^t}$. Ya que $s = k^{-1}(z + rd_A)$, el atacante ahora puede calcular la clave privada $d_A = \frac{sk-z}{r}$.

Para asegurar que k sea única para cada mensaje, uno puede omitir completamente la generación de números aleatorios y generar firmas deterministas derivando k tanto del mensaje como de la clave privada. [40]

4.6.4. Algoritmo de Verificación de Firma

Para que el destinatario autentique la firma del remitente, debe tener una copia de su punto de curva de la clave pública Q_A . El remitente puede verificar que Q_A es un punto de la curva válido de la siguiente manera:

1. Verificar que Q_A no es igual al elemento de identidad O , y sus coordenadas en todo caso sean válidas.
2. Verificar que Q_A esté en la curva
3. Verificar que $n \times Q_A = O$.

Una vez habiendo seguido estos pasos:

1. Verificar que r y s sean enteros en $[1, n - 1]$. En caso de que no lo sean, la firma es inválida.
2. Calcular $e = \text{HASH}(m)$, donde HASH es la misma función utilizada para la generación de la firma
3. Dejar z ser el L_n bits más a la izquierda de e
4. Calcular $u_1 = z^{-1} \text{ mód } n$ y $u_2 = rs^{-1} \text{ mód } n$
5. Calcular el punto de curva $x_1, y_1 = u_1 \times G + u_2 \times Q_A$. Si $x_1, y_1 = o$ entonces la firma es inválida.
6. La firma es válida si $r \equiv x_1 \text{ (mód } n)$. De otra forma es inválida.

Cabe aclarar que una implementación eficiente calcularía la inversa $s^{-1} \text{ mód } n$ solamente una vez. Además, utilizando una suma de dos multiplicaciones escalares $u_1 \times G + u_2 \times Q_A$ puede realizarse el cálculo de una manera más rápida que dos multiplicaciones escalares hechas de manera independiente. [26]

A continuación, se muestra la implementación de ECDSA en MicroPython.

4.7. Implementación ECDSA MicroPython

La documentación detallada que explique el funcionamiento de cada una de las funciones creadas para la librería de ECDSA en MicroPython que utilizamos para firmar los datos desde la ESP32 y verificarlos en el centro de control, puede ser consultada en el repositorio de este proyecto [2], en la sección de documentación, esta sección busca explicar los pasos seguidos dentro del código para firmar los datos sin incluir secciones de código ya que estas podrían incrementar considerablemente el tamaño del presente documento e incluso confundir al lector sobre el funcionamiento del algoritmo de forma meramente teórica. Se implementaron las siguientes funciones en la librería para el funcionamiento correcto del ECDSA.

- Para calcular inversos multiplicativos: Se implementaron dos funciones `calc_inv()` y `exp_binaria()`, la primera función da uso del algoritmo extendido de euclides (también implementado dentro de la librería) para hallar el inverso multiplicativo de un número k en un campo Zn , cabe mencionar que n en este caso puede o no ser primo, mientras que la función `exp_binaria()` es capaz de encontrar el inverso multiplicativo de forma rápida, sin embargo n tiene que ser primo, de lo contrario el algoritmo no funcionará de forma correcta.
- Para calcular inversos aditivos: Se implementó la función `inv_aditivo()` que toma a un entero x en un campo Zn y encuentra x^{-1} , La suma $x + x^{-1}$ tendrá que dar como resultado el elemento neutro de la suma, en este caso el 0.
- Función para calcular la pendiente: la función `pendiente()` calcula la pendiente entre dos puntos dependiendo si los puntos en la curva son iguales en coordenadas o diferentes. Se tienen tres casos:

- $P=Q$: En este caso se calcula la pendiente de la siguiente manera:

$$s = \frac{3x_1^2 + a}{2y_1} \quad (1)$$

- $P \neq Q$ y ninguno es elemento neutro: En este caso se calcula la pendiente de la siguiente manera:

$$s = \frac{y_2 - y_1}{x_2 - x_1} \quad (2)$$

- $P \neq Q$ y uno es el elemento neutro: Aquí simplemente se regresa el elemento no neutro.

Las fórmulas para calcular la pendiente pueden ser consultadas aquí [35]. Cabe mencionar que las operaciones de pendiente son sobre un módulo p , el cual se define por la curva elíptica en la que está trabajando.

- Función para sumar dos puntos de la curva: La función `suma_de_la_curva()` va a necesitar como parámetro a la pendiente de la curva, por lo que primero se debe de calcular esta. Una vez

calculada la pendiente, se calcula (x_3, y_3) , lo cual es el resultado de la suma de dos puntos. Se calcula esto según las siguientes ecuaciones:

$$x_3 = s^2 - x_1 - x_2 \quad (3)$$

$$y_3 = s(x_1 - x_3) - y_1 \quad (4)$$

De igual manera, las fórmulas para calcular la suma de puntos pueden ser consultadas aquí [35]. Los resultados de las sumas son sobre el respectivo módulo de la curva elíptica definida.

- Función de multiplicación de escalar por punto de la curva: La función `mult_binaria` retorna la multiplicación de un punto de una curva elíptica $G=(x,y)$ predefinida por los parámetros a y p , por un escalar n , de tal forma que:

$$nG = \sum_1^n G \quad (5)$$

La cualidad de esta función es que la computación no va a tener que ser generada n veces, sino que se disminuirá el número de operaciones necesarias para calcular el resultado de acuerdo al algoritmo `double and add`. El funcionamiento del algoritmo puede consultarse aquí [1].

- Funciones del tipo `get`: Funciones como `get_private_key()` o `get_gx()` definen los parámetros de la curva NIST P-256 a utilizar. Los parámetros pueden ser consultados aquí [5].
- Función de generación de firma: Se creó la función `sign_message()` que recibe como parámetros de entrada el mensaje a firmar y la clave privada a utilizar. Una vez definidos los parámetros de la curva y el mensaje, se siguen los siguientes pasos para la generación de la firma[35]:

1. Se define una llave privada x de forma pseudo-aleatoria entre 1 y $n - 1$, donde n es el número de elementos que cumplen con la condición de congruencia

$$y^2 \equiv x^3 + ax + b \pmod{p} \quad (6)$$

2. Se computa la llave pública xG , donde G es el punto generador de la curva, en nuestro caso G se definió de acuerdo a la curva P-256.

$$Ux, Uy = x(Gx, Gy) \quad (7)$$

3. Se define k de forma pseudo-aleatoria entre 1 y $n - 1$.

4. Se computa la llave pública kG , de tal forma que:

$$x_1, y_1 = k(Gx, Gy) \quad (8)$$

5. Se calcula $r = x_1 \bmod n$
6. En caso de que $r == 0$ se regresa al paso número 3.
7. Se calcula $k^{-1} \bmod n$ con la función `calc.inv()` o con exponenciación binaria.
8. Se calcula el hash del mensaje a encriptar, en nuestro caso con el algoritmo SHA-256, después pasamos el resultado de hexadecimal a un entero.
9. Se calcula $s = k^{-1}(e + xr) \bmod n$.
10. En caso de que $s == 0$ se regresa al paso número 3.
11. En caso de que $s \neq 0$ se regresa r y s.

La función de verificación de firma se implementa en el centro de control, por lo que la explicación del funcionamiento de esta será discutida en las secciones que traten acerca del funcionamiento del centro de control.

4.8. Montado del Centro de Control

Para nuestro caso el centro de control tendrá la función principal de tomar los datos de la segunda base de datos y verificar la firma que contienen para así saber si los datos son auténticos o no, primeramente el centro de control hará un request a la segunda base de datos para descargarlos, una vez descargados el centro de control se encargará de correr la función llamada `verify_ecdsa_signature()` que funciona de la siguiente manera.

Se toman como parámetros r y s, los cuales deben de estar en un intervalo de $[1, n-1]$, además de un mensaje de prueba m en forma de string, así como también otros parámetros que ya se habían visto anteriormente como G, que representa la multiplicación de un punto de la curva elíptica, así como otros parámetros previamente mencionados como n, a, etc. Para esta función, lo primero que se hace es que se debe de revisar si r y s efectivamente se encuentran en el intervalo mencionado, esto por medio de un condicional if para poder confirmar esto. En caso de que no, la función va a regresar un mensaje notificando al usuario que está fallando esta cuestión. Posteriormente, se hashea el mensaje m usando SHA-256 y se convierte a formato decimal. Después, se manda a llamar la función `calc.inv()` con el fin de utilizarla para calcular el inverso multiplicativo de s y usando el parámetro n para calcular la parte de módulo n. Asimismo, se calcula después u1 y u2 utilizando algunos de estos parámetros mencionados, como r y n por ejemplo, así como la variable w, que representa el inverso multiplicativo que se calculó anteriormente, haciendo uso de los parámetros s y n. Tras esto, se calculan x_1, y_1 y x_2, y_2 mediante la función `mult_binaria()`, empleándose para la multiplicación binaria como se explicó anteriormente. Asimismo, se usa también la función para suma de la curva para calcular x_3 y y_3 . Después, se verifica si estas coordenadas equivalen a 0,0 para, en caso de que sea así, imprimir un mensaje notificando al usuario que se rechaza la firma.

Finalmente, se usa nuevamente el parámetro n para calcular x_3 módulo n y se mete esto en una variable para posteriormente aceptar la firma, lo cual se hace de manera válida solamente si esta variable en cuestión resulta ser igual que el parámetro r .

Una vez implementado el algoritmo de verificación de firma, se procede a hacer un request a la base de datos que contiene los paquetes de datos con sus firmas y hash. El formato de la segunda base de datos se muestra en la Figura 10, como ejemplo.

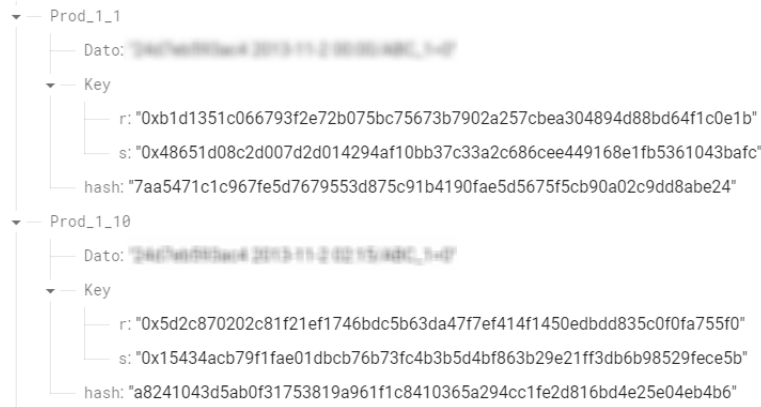


Figura 10: Ejemplo de la estructura general de la segunda base de datos, en formato json.

Una vez obtenidos los datos mediante el request, el centro de control correrá la función `verify_ecdsa_signature()` para cada uno de los datos disponibles. El output a partir de este código debería de ser como el de la Figura 11.

```

Prod_1_1 status: Verified
Prod_1_10 status: Verified
Prod_1_11 status: Verified
Prod_1_12 status: Verified
Prod_1_13 status: Verified
Prod_1_14 status: Verified
Prod_1_15 status: Verified
Prod_1_16 status: Verified
Prod_1_17 status: Verified
Prod_1_18 status: Verified

```

Figura 11: Output generado desde el centro de control, este indica si los paquetes de datos son auténticos o no.

De igual manera, si se quisiera interrumpir la ejecución de procesamiento de datos del auditor podría emplearse una herramienta como Arduino Cloud [60]. Para este primer modelo no se implementó de forma integrada esta característica debido a falta de tiempo, sin embargo sí se investigó

el cómo realizar una conexión entre la tarjeta ESP32 y Arduino Cloud para tener algún tipo de comunicación directa entre ambos puntos. A continuación se muestran los pasos a seguir para crear en primera instancia una comunicación exitosa entre la ESP32 y Arduino Cloud. En nuestro caso, una vez que la comunicación es exitosa, se cuenta con un switch para prender o apagar el led integrado a la tarjeta dentro de la interfaz de Arduino Cloud. Faltaría solo ver cómo enlazar este switch con una variable que pueda interrumpir el programa dentro de un bloque de código en C que se ejecute en MicroPython. Documentación acerca de cómo ejecutar código de C desde MicroPython puede ser encontrada aquí [12]. A continuación, se muestran los pasos para crear una conexión directa entre ambos puntos.

4.8.1. Conexión entre ESP32 y Arduino Cloud

Cabe aclarar que, dado que se cuenta solamente con una placa ESP32 en físico, entonces solamente se utilizó este único dispositivo en vez de los dos de manera simultánea como se quería llevar a cabo en un planteamiento original. Tomando esto en cuenta, se procedió a realizar los siguientes pasos para conectar esta mencionada placa ESP32 con Arduino Cloud:

1. Crear cuenta de IoT Cloud. Esto se realizó desde la pagina de Sign Up de Arduino [4].
2. Conectar placa a la computadora donde se estuviera usando Arduino Cloud (primeramente para conectar por WiFi a la placa, después no tiene que ser requerimiento usar la placa en la misma computadora que Arduino Cloud).
3. Crear "Thing" dentro de IoT Cloud y asociar un dispositivo del tipo "DOIT ESP32 DEVKIT V1". Una vez creado IoT Cloud proporciona un Device ID y un Secret Key para este dispositivo, como se muestra en la Figura 12:

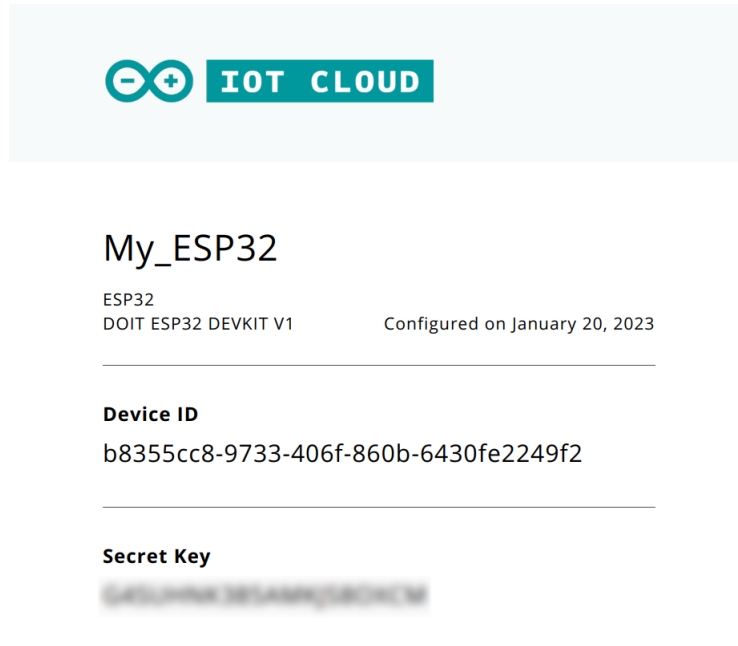


Figura 12: Credenciales generadas para conectar Arduino Cloud con ESP32.

4. Conectar ESP32 con alguna red 2.4; para esto se requirió de las credenciales de una red 2.4, así como de la llave secreta generada para la placa ESP32, tal como se muestra en la Figura 13:

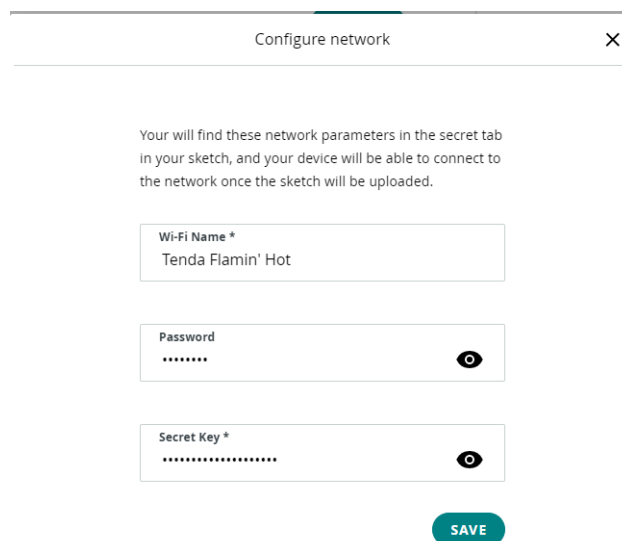


Figura 13: Credenciales generadas para conectar Arduino Cloud con ESP32.

5. Una vez realizada la conexión, se deberá de ver el siguiente recuadro de información desde la sección de Setup dentro de "Things". Véase la Figura 14:



Figura 14: Configuración de red a utilizar dentro de la ESP32

6. Para verificar la comunicación correcta entre ESP32 y Arduino Cloud, se procedió a realizar un script que activa ciertas funciones del ESP32 desde Arduino Cloud, en este caso un switch para activar y desactivar su led integrado y un contador que se actualiza cada 5 segundos de forma automática. Ambas funciones se realizaron de forma exitosa. El script fue realizado desde la sección "Sketch" dentro de Arduino Cloud y puede ser consultado aquí [3].

A continuación, en la Figura 15 se muestra como ejemplo de la conexión exitosa entre la placa ESP32 y Arduino Cloud, el led integrado de la placa encendido:

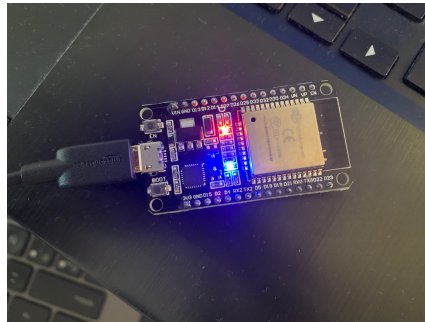


Figura 15: Led Encendida en la ESP32 de forma exitosa (led azul).

4.9. Creación de Certificados

Para la creación de certificado se adapto la librería de X.509 de cryptography [56] utilizando las funciones del mismo modulo para poder crear un documento .crt para cada auditor, dándole su

respectivo nombre, su fecha de inicio y expiración y sus llaves, las cuales se crearon utilizando la librería `cryptography`, también utilizamos la librería de `hashlib` para crear los hashes de los documentos y agregar una pared de seguridad. Por motivos de tiempo no alcanzamos a implementar los certificados `.crt` al auditor, sin embargo estos se crean de forma correcta, por lo que se cree que no debería pasar a mayores problemas de implementaciones si se trabajara en futuras versiones. El código que implementa los algoritmos de certificados puede ser encontrado dentro de nuestro repositorio [2], dentro de la carpeta de centro de control que se encuentra en la carpeta del reto.

5. Vectores de Prueba

En nuestro repositorio [2], se tiene una sección que incluye un jupyter notebook de prueba que muestra el funcionamiento correcto del algoritmo de ECDSA implementado. Primeramente se definen los parámetros de la curva NIST P-256, una llave privada y un valor `k` predefinidos con respecto al ejemplo que se encuentra en el RFC-6979, sección A.2.5. aquí [54].

Primeramente, se verifica que la operación `xG` genere las siguientes llaves públicas:

$$U_x : 60fed4ba255a9d31c961eb74c6356d68c049b8923b61fa6ce669622e60f29fb6 \quad (9)$$

$$U_y : 7903fe1008b8bc99a41ae9e95628bc64f2f1b20c2d7e9f5177a3c294d4462299 \quad (10)$$

Al realizar la operación de escalar por punto generador desde nuestra función `mult_binaria()` se obtiene el siguiente output, el cual se puede ver en la Figura 16:

```
Ux: 0x60fed4ba255a9d31c961eb74c6356d68c049b8923b61fa6ce669622e60f29fb6
Uy: 0x7903fe1008b8bc99a41ae9e95628bc64f2f1b20c2d7e9f5177a3c294d4462299
```

Figura 16: Resultado de la operación escalar por punto generador implementada desde la librería ECDSA signature hecha por el equipo.

Se puede ver que los resultados obtenidos en la Figura 11 son congruentes con los que se deberían de obtener para la llave pública desde el RFC-6979.

Ya verificada la clave pública, se firma el mensaje *sample*, con los parámetros antes mencionados y con `k = 0xA6E3C57DD01ABE90086538398355DD4C3B17AA873382B0F24D6129493D8AAD60`

Según el RFC-6979, se deberían de obtener los siguientes resultados para las siguientes variables:

$$r : efd48b2aacb6a8fd1140dd9cd45e81d69d2c877b56aaf991c34d0ea84eaf3716 \quad (11)$$

$$s : f7cb1c942d657c41d436c7a1b6e29f65f3e900dbb9aff4064dc4ab2f843acda8 \quad (12)$$

Al realizar nuestro algoritmo de firmado se obtuvo el siguiente output, el cual se puede ver en la Figura 17:

```
r: 0xefd48b2aacb6a8fd1140dd9cd45e81d69d2c877b56aaf991c34d0ea84eaf3716
s: 0xf7cb1c942d657c41d436c7a1b6e29f65f3e900dbb9aff4064dc4ab2f843acda8
```

Figura 17: (r, s) generados a partir de nuestro algoritmo de firmado.

De igual manera, se puede ver que los parámetros (r, s) son congruentes con los que se obtienen en el RFC-6979.

Por último, se verifica la autenticidad del mensaje por medio de la función `verify_ecdsa.signature()`, que toma como parámetros a la llave pública, el mensaje original y a los anteriormente generados (r, s). En caso de que la función retorne `False`, se sabrá que el dato no fue autenticado de forma correcta y que pudo haber sido modificado, alterado o manipulado en alguno de sus parámetros.

En nuestro caso, la función arroja `True`, lo cual indica que los datos son auténticos y que la verificación de firmas fue realizada de forma correcta, si se quieren comprobar los resultados, se puede acceder en el repositorio del proyecto [2] al notebook titulado *Ejemplo_ECDSA_RFC6979.ipynb*.

Ahora bien, para probar el hash con SHA-256, se emplea a modo de prueba un mensaje `m` denominado como "sample". Para ello se utilizó el siguiente fragmento de código en Python:

```
1 m = "sample"
2 hash_object = hashlib.sha256()
3 hash_object.update(m.encode())
4 hexa = binascii.hexlify(hash_object.digest())
5 e = int(hexa, 16)
6 print("Hash con SHA-256:", hexa)
```

Con este caso de prueba, se obtiene lo siguiente: Hash con SHA-256:
b'af2bdbelaa9b6ec1e2ade1d694f41fc71a831d0268e9891562113d8a62add1bf'

Asimismo, se toma otro caso de prueba, probando ahora con un mensaje `m` denominado como "prueba", con lo cual, tomando como base el fragmento de código anterior, se obtiene lo siguiente: Hash con SHA-256: b'655e786674d9d3e77bc05ed1de37b4b6bc89f788829f9f3c679e7687b410c89b'

De igual manera, en el siguiente caso de prueba, probando con un mensaje `m` denominado como "test", y basándose en el código mencionado, se obtiene lo siguiente: Hash con SHA-256: b'9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08'

Finalmente, en un caso de prueba adicional mostrado a continuación, se prueba con un mensaje *m* denominado como "*ejemplo*", con lo cual, tras probarlo con el código, se obtiene: Hash con SHA-256: b'fd984bad363e9a30021460091726467b65bb1e88c31752d93c68ade87e4946e4'

6. Resultados

Analizando los resultados obtenidos, se puede observar que la conexión que se creó entre la primera base de datos y el auditor es efectiva, puesto a que dicha conexión funciona de la forma deseada. Se puede decir esto porque no se encuentra la necesidad de cambiar la primera base de datos desde el auditor. Es importante comentar que las conexiones entre bases de datos y dispositivos son del tipo https, lo cual se encarga de cubrir el apartado de mantener la confidencialidad de los datos con los que se está trabajando. Por otro lado, se logró verificar con el RFC-69-79 [54] que el hasheo de los datos y que la implementación del propio algoritmo de ECDSA lograran sus cometidos de manera efectiva. En cuanto al centro de control, se consiguió que este pudiera ser capaz de correctamente recibir y procesar datos. Recordando lo visto en la Figura 11 del documento, los tiempos de computación dentro del micro procesador varían en lapsos de tiempo de alrededor de 15 segundos para que este pueda recibir, transformar y enviar un paquete de datos. Por lo tanto, se puede decir que esto es un resultado bastante óptimo, ya que considerando la situación problema, la medición y envío de trazas se realizan cada 15 minutos. En cuanto a la verificación, por cada dato dentro del centro de control es menor a 1 segundo, específicamente de 0.18 segundos en promedio. Lógicamente, dicho tiempo de verificado por paquete dependerá de las especificaciones técnicas del equipo de cómputo, es decir, que tan rápido se realizará el proceso de verificación de firmas. En el caso particular de este proyecto, para 154 datos de prueba usados en el código desarrollado, el script tardó 33.35 segundos en verificarlos todos, teniendo como centro de control una computadora con un procesador intel Core i7 de octava generación, una RAM instalada de 16 GB y que los programas se ejecutaron en PyCharm community edition con la versión de Python 3.11.1. Los resultados se pueden observar en la Figura 18.

```
Cons_2_57 status: Verified
Cons_2_58 status: Verified
Cons_2_6 status: Verified
Cons_2_7 status: Verified
Cons_2_8 status: Verified
Cons_2_9 status: Verified

--- 33.35289001464844 seconds ---
```

Figura 18: Tiempo de ejecución del programa de verificación de firmas dentro del centro de control.

En resumen, se puede decir que las conexiones entre base de datos, auditor y centro de control son efectivas, ya que la transferencia de datos cumple con mantener su confidencialidad, integridad y autenticidad. Dichos procesos se pueden lograr en tiempos de trabajo óptimos, aunque sí está sujeta a cambios, dependiendo de las capacidades de la computadora que aplique estos algoritmos.

7. Conclusiones

Podemos concluir que la conexión entre bases de datos de FireBase y un ESP32 puede ser efectuada de forma exitosa, también se pudo transmitir información desde la ESP32 al centro de control de forma indirecta, ya que los datos primero tenían que pasar por la segunda base de datos para que estos pudieran ser recibidos por el centro de control, esto, aun y que no es lo ideal, funciona cuando el centro de control tiene que únicamente verificar los datos que recibe, en la sub sección dentro de este apartado, se podrán encontrar algunas de las recomendaciones de uso del sistema actual y puntos de mejora en caso de que se ampliar y optimizar el funcionamiento de este sistema para ambientes con dispositivos IoT.

Se concluye que el tiempo de procesamiento de los datos es muy bueno tomando en cuenta que la mayoría del trabajo se esta realizando en un ESP32 y tomando en cuenta que en la práctica el envío de datos necesita ser hecho cada 15 minutos, claro que, tomando en cuenta que es un microchip el que realiza el trabajo, se podrían acortar los tiempos de procesamiento utilizando otros tipos de dispositivos con mayores capacidades de procesamiento, como un Raspberry Pi 4B, sin embargo el costo es mucho mayor al de un ESP32.

También se concluye que este es un sistema que no ha sido probado en situaciones reales y que por lo tanto no debería ser usado en primera instancia con datos reales que puedan ser considerados como sensibles, podrían implementarse medidas de ciberseguridad para identificar posibles fallas y puntos a mejorar en cuanto en la seguridad del sistema, como por ejemplo una simulación de Ataques y Defensas con un Red Team y un Blue Team.

Se observa también que la implementación del algoritmo de ECDSA bajo los parámetros de la curva NIST P-256 en un ESP32 es efectiva para la situación problema planteada, la seguridad de este algoritmo es muy robusto debido al tamaño de las firmas, tomando en cuenta que el tiempo de firma por dato dentro de la ESP32 es de alrededor de 15 segundos concluimos que puede ser muy útil implementar este algoritmo para situaciones donde la transmisión de datos pueda ser mayor a 15 segundos por dato, sin embargo se considera que si se tuviera que mandar repetidamente datos en intervalos menores a 15 segundos con un dispositivo con componentes similares al de una ESP32, implementar ECDSA con la curva NIST P-256 puede no ser óptimo para el sistema, por lo que se recomienda otro tipo de curvas más pequeñas como la NIST P-192, claro esto dependerá de que tan robusta se quiera que sea la firma.

Se concluye también que las herramientas que ofrece Firebase son efectivas para esta situación

en concreto y que puede haber una escalabilidad del proyecto para que pueda haber más auditores conectados a la base de datos de forma simultánea, sin embargo, hay que considerar que el aumento de almacenamiento y transmisión de datos en algún punto hará que la herramienta deje de ser gratuita, por lo que se recomienda seguir explorando diferentes alternativas que ofrezcan almacenamiento de datos en la nube, esto incluso podría llegar a no ser un requerimiento en el futuro en caso de que el auditor este conectado físicamente o en alguna red local con la base de los datos y si se puede montar la conexión directa auditor-centro de control.

Por otra parte, se tiene que tomar en cuenta la falta de conectividad directa entre el centro de control y el auditor, esto puede tener consecuencias como falta de control de forma remota en los auditores, si se quisiera parar el trabajo de un auditor o que este lea algún tipo de dato de forma rápida, se necesitaría implementar una conexión directa del tipo cliente servidor entre centro de control y auditor, más detalles de este tema en la sección de recomendaciones.

Por último se concluye que este sistema es capaz de realizar de forma efectiva el firmado y verificación de los datos, garantizando de forma exitosa la integridad y autenticidad de las trazas a enviar al centro de control, por otra parte vemos que todas las solicitudes generadas hacia Firebase son del tipo https, esto implica que la confidencialidad de los datos esta cubierta mediante el protocolo SSL (Secure Socket Layer) [13]. Se reconoce por parte de todo el equipo que puede haber mejoras, sobre todo en la integración de funcionalidades dentro del centro de control, como se mencionó en partes anteriores, el implementar certificados a los auditores que se generan dentro del centro de control y poder tener una conexión bidireccional directa entre auditor y centro de control son dos puntos importantes a cubrir para mejorar el sistema en futuras implementaciones.

7.1. Recomendaciones

A continuación, se presentan las recomendaciones para el uso del sistema actual y mejoras a implementar en el proyecto:

- Implementar un sistema de comunicación directa bidireccional entre la ESP32 (auditor) y el centro de control. En este caso se tendría que crear un servidor que pueda recibir y procesar la información de la ESP32 y poder enviar señales, se intentará implementar un sistema para interrumpir la ejecución de programas dentro de la ESP32 a partir de un Arduino Cloud, esta podría ser una forma de control la ESP32 de forma remota. Ya se mostró cómo montar una conexión entre Arduino Cloud (que podría fungir como parte del auditor) y la ESP 32 en la sección 4.8.1 del documento, pasos a seguir también se mencionan en esta sección. No fue posible implementar este sistema debido a falta de tiempo y a que como equipo, nos enfocamos más en la implementación correcta de la librería de ECDSA dentro de auditores y centro de control, así pues, se considera de gran importancia para mejorar el funcionamiento de el sistema general propuesto.
- Si se deseara acortar los tiempos de procesamiento, se podría transferir este sistema a otro micro

chip, o algún otro dispositivo con mayor poder computacional. Un dispositivo que pueda usar Python3 podría simplificar e incluso mejorar la implementación de algunos de los algoritmos hechos.

- En caso de querer hacer una conexión directa entre la ESP32 y el centro de control con protocolo http, se tendría que implementar algún algoritmo de cifrado de datos para garantizar la confidencialidad de los datos.
- En caso de querer utilizar de manera correcta los certificados, el uso adecuado seria que alguna entidad externa a la organización y el centro de control cree el certificado para el centro de control, pues la mejor practica no es que el centro de control cree su propio certificado sino alguna entidad externa. Para uso no productivo pero practico y para la funcionalidad del la solución se utilizo una manera para crear certificados el cual puede ser mejorado al usar las llaves que se crean desde un principio sin tener que utilizar RSA para crearlas.
- Implementar un sistema de archivos .log que registren toda la actividad del auditor desde el centro de control, la información que indique la actividad del auditor puede ser registrada en la segunda base de datos para que luego esta sea accesada por el centro de control.

Referencias

- [1] Elliptic curve cryptography: a gentle introduction. <https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/>. (Accessed: 02/03/2023).
- [2] Repositorio del proyecto de la materia ma2006b. <https://github.com/pablormzsantaellau/InvCripto3>. (Accessed: 02/06/2023).
- [3] Script de prueba para esp32. https://drive.google.com/file/d/1Boi6_FyCQ0LftumJx6vBoSKda2be70JN/view?usp=sharelink. (Accessed : 01/22/2023).
- [4] Sign up to arduino. <https://login.arduino.cc/login>. (Accessed: 01/22/2023).
- [5] Standard curve database. <https://neuromancer.sk/std/nist/P-256>. (Accessed: 02/03/2023).
- [6] Time access and conversions. <https://docs.python.org/3/library/time.html/>, 2018. (Accessed: 04/07/2023).
- [7] Python-firebase. <https://pypi.org/project/python-firebase/>, 2019. (Accessed: 04/07/2023).
- [8] Secure hashes and message digests. <https://docs.python.org/3/library/hashlib.html>, 2019. (Accessed: 04/07/2023).
- [9] Convert between binary and ascii. <https://docs.python.org/3/library/binascii.html>, 2020. (Accessed: 04/07/2023).
- [10] Miscellaneous operating system interfaces. <https://docs.python.org/3/library/os.html>, 2022. (Accessed: 04/07/2023).
- [11] Network basics. https://docs.micropython.org/en/latest/esp8266/tutorial/network_basics.html, 2022. (Accessed : 04/07/2023).
- [12] Micropython external c modules. <https://docs.micropython.org/en/latest/develop/cmodules.html>, 2023. (Accessed: 02/03/2023).
- [13] Privacidad y seguridad en firebase. <https://firebase.google.com/support/privacy?hl=es-419>, 2023. (Accessed: 02/07/2023).
- [14] C. 101. Nodemcu esp8266. <https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet>. (Accessed: 01/13/2023).
- [15] anukruti16. What is p2p(peer-to-peer process)? <https://www.geeksforgeeks.org/what-is-p2ppeer-to-peer-process/>, 2022. (Accessed on 09/01/2023).
- [16] ArunEworld. Esp8266 – various platform supports. <https://aruneworld.com/embedded/esp8266/esp8266-platforms/:text=ESP8266>. (Accessed: 01/13/2023).

- [17] M. Babiuch. Using the esp32 microcontroller for data processing. <https://ieeexplore.ieee.org/document/8765944>, 2019. (Accessed on 12/01/2023).
- [18] D. de Firebase. Planes de precios de firebase. <https://firebase.google.com/docs/projects/billing/firebase-pricing-plans?hl=es-419>. (Accessed: 08/12/2023).
- [19] A. DOCS. Arduio uno r3. <https://docs.arduino.cc/hardware/uno-rev3>. (Accessed: 01/13/2023).
- [20] F. DOCS. Precios de firestore. <https://cloud.google.com/firestore/pricing?hl=es-419>. (Accessed: 08/12/2023).
- [21] C. Dustar, S. Lachner. A performance evaluation of data protection mechanisms for resource constrained iot devices., 2019. (Accessed on 12/01/2023).
- [22] H. Dweik and M. Abutaha. *Lightweight Cryptographic Techniques and Cybersecurity Approaches*. Intech Open, 2022. Chapter 2, (Accessed: 01/13/2023).
- [23] Firebase. Calculadora de precios de firebase. <https://firebase.google.com/pricing?hl=es-419>. (Accessed: 08/02/2023).
- [24] I. J. B. García. ¿qué es la arquitectura de red y cuáles son sus funciones? <https://www.servnet.mx/blog/la-arquitectura-de-red-y-sus-funciones-para-un-buen-desempeno>, 2021. (Accessed on 09/01/2023).
- [25] S. Gautam. Peer to peer (p2p) architecture. <https://www.enjoyalgorithms.com/blog/peer-to-peer-networks>, 2022. (Accessed on 10/01/2023).
- [26] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.
- [27] E. Hub. Getting started with esp32. <https://www.electronicshub.org/getting-started-with-esp32/>. (Accessed: 01/13/2023).
- [28] IBM. The client/server model. <https://www.ibm.com/docs/en/zos/2.1.0?topic=applications-clientserver-model>, 2021. (Accessed on 10/01/2023).
- [29] IBM. Ibm documentacion: Opciones de seguridad de la transmision. <https://www.ibm.com/docs/es/i/7.2?topic=security-transmission-options>, abril 2021. (Accessed on 09/01/2023).
- [30] IBM. opciones de seguridad de la transmisión. <https://www.ibm.com/docs/es/i/7.2?topic=security-transmission-options>, 2021. (Accessed: 01/14/2023).
- [31] Instructure. Reto. <https://experiencia21.tec.mx/courses/344009/pages/reto>, 2023.

- [32] IntelliPaat. What is client server architecture? <https://intellipaas.com/blog/what-is-client-server-architecture/>, 2022. (Accessed: 01/16/2023).
- [33] J. Inza. Root ecc para la internet de las cosas (iot). <https://inza.wordpress.com/2016/05/10/root-ecc-para-la-internet-de-las-cosas-iot/>, 2016.
- [34] iobotic. Como instalar micropython en la esp32 - fÁcil - v1. <https://www.youtube.com/watch?v=RLqPB1PM6gElist=PLtZQ6WyfEW8MqUpt2VmDr5FuJe3CiV7bA>. (Accessed: 01/22/2023).
- [35] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.
- [36] KeepCoding. ¿qué es la criptografía de curva elíptica? <https://keepcoding.io/blog/que-es-la-criptografia-de-curva-eliptica/>, 2022.
- [37] A. Khalique, K. Singh, and S. Sood. Implementation of elliptic curve digital signature algorithm. *International journal of computer applications*, 2(2):21–27, 2010.
- [38] V. Kolic, E. Spaho, K. Matsuo, S. Caballe, L. Barolli, and F. Xhafa. Implementation of a medical support system considering p2p and iot technologies. In *2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems*, pages 101–106, 2014.
- [39] J. Maldonado. Criptografía de curva elíptica (ecc), el corazón de la seguridad en el mundo cripto y de internet. <https://es.cointelegraph.com/explained/elliptic-curve-cryptography-ecc-the-heart-of-security-in-the-crypto-and-internet-world>, 2020.
- [40] S. Matheu. Curvas elípticas. https://www.um.es/documents/118351/1884002/TFG_MATHEU+GARCIA.pdf/0f3f6eb9-5ef7-4483-b41f-525bf7ef1160, 2015.
- [41] N.A. Digital signature service (dss). <https://dss.eadtrust.eu/>.
- [42] N.A. Eadtrust. <https://www.incibe.es/protege-tu-empresa/catalogo-de-ciberseguridad/listado-empresas/eadtrust>.
- [43] N.A. Las redes p2p. <https://www.andaluciaesdigital.es/educarparaproteger/adolescentes/capitulos/perfilestics/redes-p2p.html>.
- [44] N.A. Qué es xampp usos, características, opiniones, precios. <https://mundobytes.com/xampp/>.
- [45] N.A. Skipjack. <https://www.hypr.com/security-encyclopedia/skipjack>.
- [46] N.A. Seguridad criptográfica en iot (ii). <https://empresas.blogthinkbig.com/seguridad-criptografica-en-iot-ii/>, 2016.

- [47] N.A. ¿qué es el algoritmo de firma ecdsa? <https://academy.bit2me.com/que-es-ecdsa-curva-eliptica/>, 2019.
- [48] N.A. Ead trust. <https://ametic.es/asociado/ead-trust/>, 2022.
- [49] N.A. ¿qué es iis (internet information services) y cómo funciona? <https://kryptonsolid.com/que-es-iis-internet-information-services-y-como-funciona/>, 2022.
- [50] N.A. ¿qué es la criptografía de curva elíptica? https://keepcoding.io/blog/que-es-la-criptografia-de-curva-eliptica/Como_s_e_u_s_a_n_i_a_s_c_u_r_v_a_s_e_l_i_p_t_i_c_a_s_e_n_c_r_i_p_t_o_g_r_a_f_i_a, 2022.
- [51] nabto:online. The nabto iot connectivity solution for device control. <https://www.nabto.com/solution/>, 2023. (Accessed on 12/01/2023).
- [52] R. Pi. Raspberry pi pico. <https://www.raspberrypi.com/products/raspberry-pi-pico/>. (Accessed: 01/13/2023).
- [53] G. C. Platform. Cloud platform skus. <https://cloud.google.com/skus/?hl=es-419>. (Accessed: 08/12/2023).
- [54] T. Pornin. Deterministic usage of the digital signature algorithm (dsa) and elliptic curve digital signature algorithm (ecdsa). Technical report, 2013.
- [55] K. Prasetyo, Y. Purwanto, and D. Darlis. An implementation of data encryption for internet of things using blowfish algorithm on fpga. https://www.researchgate.net/publication/272157650_An_implementation_of_data_encryption_for_Internet_of_Things_using_blowfish_algorithm_on_FPGA, 2014.
- [56] PYCA. X.509 cryptography. <https://cryptography.io/en/latest/x509/>. (Accessed: 02/03/2023).
- [57] M. Pérez. ¿qué es firebase? <https://www.iebschool.com/blog/firebase-que-es-para-que-sirve-la-plataforma-desarrolladores-google-seo-sem/>, 2016.
- [58] S. e. a. Singh. Advanced lightweight encryption algorithms for iot devices: survey, challenges and solutions. *Journal of Ambient Intelligence and Humanized Computing*, 2017.
- [59] SSL. ¿qué es la criptografía de curva elíptica (ecc)? <https://www.ssl.com/es/preguntas-frecuentes/que-es-la-criptografia-de-curva-eliptica/>, 2021.
- [60] K. Söderby. Getting started with the arduino iot cloud. <https://docs.arduino.cc/arduino-cloud/getting-started/iot-cloud-getting-started>, 01 2023. (Accessed: 01/19/2023).
- [61] Wokwi. Raspberry pi pico. <https://wokwi.com/projects/new/micropython-pi-pico>, 2022. (Accessed: 01/14/2023).

[62] Wokwi. Welcome to wokwi! https://docs.wokwi.com/?utm_source=wokwi, 2022. (*Accessed* : 01/14/2023).

Los créditos de las fotografías pertenecen a sus respectivos autores.