



## Uso de álgebras modernas para seguridad y criptografía

Implementación de criptografía de clave pública para protección de comunicaciones y almacenamiento de datos con IoT en entornos de monitoreo y consumo de energía

### **Autores:**

Jose Alfredo García Rodríguez | A00830952  
Daniel De Zamacona Madero | A01570576  
Verónica Victoria García De la Fuente | A00830383  
Jose Miguel Perez Flores | A00832401  
Karla Susana Olvera Vázquez | A01379097  
Eugenio Santisteban Zolezzi | A01720932

### **Profesor:**

Alberto F. Martínez

20 de junio de 2023

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Estado del arte</b>	<b>4</b>
2.1. Revisión de arquitecturas de intercambio de información en red (P2P, Blockchain, cliente-servidor, etc).	4
2.1.1. Red P2P	4
2.1.2. Blockchain	5
2.1.3. Cliente-servidor	5
2.2. Aplicaciones de arquitectura en IoT	6
2.3. Recursos disponibles	6
2.4. Generador de credenciales y su registro en un servidor AAA	7
2.5. Protocolos posibles para IoT	7
2.6. Criptografía de clave pública	8
2.7. Bibliotecas disponibles para acelerar la implementación de los esquemas de clave pública	9
2.8. Virtualización de la red	9
<b>3. Algoritmos de encriptado posibles</b>	<b>10</b>
3.1. RSA	10
3.2. ECC	10
3.3. ECDSA	10
<b>4. Implementaciones similares en la bibliografía</b>	<b>11</b>
<b>5. Descripción y componentes de la propuesta</b>	<b>11</b>
5.1. Seguridad física	12
5.2. Certificados	13
5.3. Encriptado de los datos	14
5.4. Firma de los datos	15
5.5. Bróker Mosquitto	16
5.6. Almacenamiento de los datos	17
5.6.1. Costos de implementación aproximados	17
<b>6. Metodología y desarrollo de la solución seleccionada</b>	<b>17</b>
6.1. Creación y firma de certificados	18
6.1.1. Certificados Centro de Control	18

6.1.2. Certificados bróker . . . . .	18
6.1.3. Certificados auditores . . . . .	19
6.2. Instalar y configurar Mosquitto . . . . .	20
6.2.1. Conexión por bróker MQTT . . . . .	24
6.3. Credenciales y archivos iniciales . . . . .	25
6.4. Encriptado, intercambio seguro de claves y firmado de las trazas . . . . .	26
6.5. Creación de base de datos e inserción de datos . . . . .	28
<b>7. Conclusiones y trabajo futuro</b>	<b>28</b>
<b>8. Anexos</b>	<b>30</b>
8.1. Vectores de Prueba . . . . .	30

# 1. Introducción

En este proyecto, nos enfocaremos en la implementación de esquemas criptográficos para proteger las comunicaciones y el almacenamiento de datos en entornos de Internet de las Cosas utilizados para monitorear la producción y el consumo de energía. El escenario planteado involucra a un grupo de auditores que tienen la capacidad de recopilar datos de dispositivos inteligentes, como medidores e inversores, ubicados en diversos entornos, como residencias, comercios, instituciones educativas e instalaciones industriales equipadas con sistemas fotovoltaicos. Estos datos son enviados a un centro de control encargado de analizar la información recopilada. Nuestro objetivo es proponer un esquema de intercambio de datos que garantice la seguridad de la información al salvaguardar la confidencialidad, integridad, autenticidad y privacidad de los datos. Dado que los patrones de consumo y producción son considerados datos sensibles y no deben ser expuestos, resulta fundamental implementar un protocolo criptográfico seguro para garantizar que los datos no sean comprometidos en ningún momento, desde su generación hasta su almacenamiento. Esta propuesta incluye no solo los algoritmos de encriptado, sino también la arquitectura de red y el almacenamiento de estos.

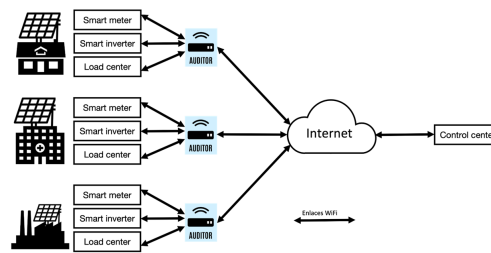


Figura 1: Diagrama arquitectura real del proyecto

La figura 1 muestra los dispositivos del smart grid encargados de monitorear los parámetros de energía obtenidos de los medidores e inversores. Estos dispositivos cuentan con una interfaz WiFi para comunicarse con el exterior, y para fines de la simulación y el entorno de pruebas correspondientes al reto, los datos comunicados son proporcionados por la OSF en un archivo .csv lo que implica que los medidores e inversores no estarán considerados como componentes de la implementación por motivos prácticos. En este escenario, los auditores digitales que recolectarán datos de los dispositivos smart conectados a ellos serán micro-controladores con capacidad WiFi. El centro de control será una computadora con el sistema operativo Ubuntu que será capaz de consultar los datos recolectados y enviados por cada uno de los auditores.

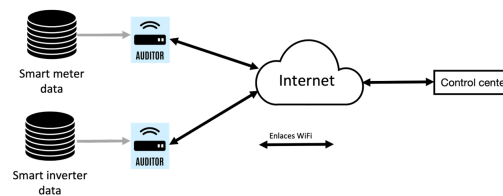


Figura 2: Diagrama arquitectura simplificada del proyecto

Para fines de simplicidad, la figura 2 muestra el esquema del sistema a manera de prueba de concepto en el que los sensores son sustituidos por los archivos antes mencionados que contienen los datos de los smart meters.

## 2. Estado del arte

### 2.1. Revisión de arquitecturas de intercambio de información en red (P2P, Blockchain, cliente-servidor, etc).

La arquitectura de red es un medio para implementar un conjunto coordinado de productos que se puedan interconectar. La arquitectura es el plan con el que se conectan los protocolos y otros programas de software para apoyar a los usuarios de la red como par a los proveedores de hardware y software. Estos están compuestos por conjuntos de equipos de transmisión, programas, protocolos de comunicación y una infraestructura radioeléctrica que hace posible la conexión y transmisión de datos a través de la red, persiguiendo un ambiente escalable, con la correcta gestión de red y respuesta ante fallos, y seguridad para los flujos de tráfico presentes. [1]

Al crear una arquitectura de red, se aumenta la seguridad de todos los integrantes y el rendimiento de los componentes, pero la orientación de los mismos varia, por lo mismo se presentan los distintos modelos de arquitecturas presentes:

- Topológicos - son simples y distribuyen los ordenadores y componentes basándose en una determinada área geográfica, aquí entran los modelos core que se encargan mayormente del trabajo en la entrada de red, con ejemplos de modelos siendo LAN, MAN, y WAN.
- Basados en el flujo de datos - estudia la relación entre dos ordenadores de la misma red, y analizando la red P2P de punto a punto y jerarquía que se tiene entre un cliente y el servidor, intenta arreglar el máximo de todos los servicios que puedan aumentar el flujo de paquetes de datos entre los componentes.
- Funcionales - mejoran las funciones de servicio que ya existen entre los diferentes niveles de la red, se ocupan de la privacidad y seguridad, de la misma forma que manejan todos los requerimientos para mejor analizar el flujo de datos.
- Combinados - Es una fusión de todos o algunos de los modelos anteriores, enriquecen funciones y presentan grandes beneficios en el flujo de paquetes de datos y la distribución geográfica.

#### 2.1.1. Red P2P

En la arquitectura de red peer-to-peer (P2P), los nodos pares de la distribución pueden actuar tanto de cliente como servidor, estas redes admiten el intercambio de información en cualquier formato entre equipos interconectados, estas redes funcionan como redes "superpuestas" que son redes lógicas o virtuales que se crean por encima de redes existentes con el fin de implementar un servicio de red que no esta disponible en la red actual, existen redes P2P estructuradas, no estructuradas e híbridas, las cuales varían en el establecimiento de los pares y sus claves de asignación [2].

##### Ventajas de una red P2P

- No necesita un servidor dedicado por lo que cuesta menos.
- Si una computadora detiene su función, otras conectadas a la red continúan funcionando.
- Instalación y configuración de la misma requiere un esfuerzo mínimo por el apoyo pre instalado en los sistema operativos.

##### Desventajas de una red P2P

- La seguridad y respaldo de datos debe hacerse en cada computadora individual.
- Mientras incrementen las computadoras en la red, el manejo de la seguridad, rendimiento y acceso se complican.

### 2.1.2. Blockchain

Es una cadena de bloques que contienen una base de datos asegurada mediante el agrupamiento en la red (P2P), es una combinación de computadoras ligadas entre ellas en vez de un servidor centralizado, resultando en una red descentralizada, y esta puede variar entre estructuras siendo publicas, privadas y una de consorcio. [3]

Dentro del escenario presente se intercambian datos entre múltiples ubicaciones, por lo mismo el encriptado de estos es vital para formar un sistema seguro y comprensible, para esto es necesario aplicar un cifrado de sus claves, estas son nuestras llaves para intercambiar formatos de la información en cuestión. En nuestra implementación contamos con esquemas de clave pública, esta también conocida como la criptografía asimétrica, utiliza ambas claves públicas y privadas de forma continua, donde cada participante tiene un par de claves y ambas se mantienen en conjunto dentro de un algoritmo matemático que las vincula, de manera que los datos que se cifran con la clave pública solo pueden descifrarse con la clave privada. Este principio de unidireccionalidad conforma todo el criptosistema asimétrico, el destinatario comunica la clave pública a la otra parte y se guarda la clave privada, el remitente codifica su mensaje con esta clave y puede enviarla al destinatario y solo sera descifrable con la clave privada [4].

La siguiente tabla muestra las similitudes y diferencias entre distintas aplicaciones de modelos de cadenas de bloques.

Propiedad	Público	Consortio	Privado
Determinación por consenso	Todos los mineros	Grupo de nodos seleccionado	Una organización
Leer permisos	Público	Público o restringido	Público o restringido
Inmutabilidad	Prácticamente imposible	Puede ser alterado	Puede ser alterado
Eficiencia	Baja	Alta	Alta
Centralizado	No	Parcial	Si
Procesamiento por consenso	Sin permisos	Con permisos	Con permisos

### 2.1.3. Cliente-servidor

Es una arquitectura de red computacional donde múltiples clientes (o procesadores remotos) solicitan y reciben servicios de un servidor centralizado (ordenador anfitrión), en esta red un servidor centralizado y poderoso actúa como el centro donde los otros ordenadores o clientes pueden conectarse. Este servidor es el corazón de cualquier solicitud realizadas por los clientes interconectados [4].

#### Ventajas de una red cliente-servidor

- Los recursos y seguridad de los datos son controlados a través del servidor.
- No esta restringida a un bajo número de ordenadores.
- Un servidor puede ser accedido a través de múltiples plataformas.

#### Desventajas de una red cliente-servidor

- Puede ser muy costoso por la necesidad de mantener el servidor y los dispositivos de la red como hubs, routers y switches.
- Cuando el servidor es afectado, la red entera será afectada.
- El apoyo técnico necesita constantemente mantener y asegurar las funciones de la red de manera eficiente.

## 2.2. Aplicaciones de arquitectura en IoT

La arquitectura cliente-servidor es ampliamente utilizada en la industria de IoT (Internet de las cosas) para permitir que los dispositivos IoT se comuniquen y procesen datos de manera eficiente. Esta arquitectura consta de dos componentes principales: un cliente y un servidor.

El cliente se refiere a cualquier dispositivo IoT que recopila datos y los envía al servidor para su procesamiento. Esto puede incluir sensores, dispositivos portátiles, automóviles, electrodomésticos inteligentes y otros dispositivos conectados a Internet. Por otro lado, el servidor es una computadora o un conjunto de computadoras que procesan los datos recibidos por el cliente y proporcionan servicios a los clientes, como almacenamiento de datos, análisis y visualización de datos.

El uso de la arquitectura cliente-servidor en IoT ha permitido que los dispositivos IoT sean más inteligentes y eficientes en el procesamiento de datos. Esto se debe a que los datos se pueden procesar en el servidor, lo que reduce la carga de trabajo en los dispositivos IoT y mejora el rendimiento general del sistema.

En cuanto a los pronósticos futuros del uso de la arquitectura cliente-servidor en IoT, se espera que su uso siga creciendo en los próximos años a medida que se desarrollen más dispositivos IoT y se integren en nuestras vidas diarias. También se espera que la arquitectura evolucione para manejar grandes volúmenes de datos y aumentar la seguridad en la comunicación entre dispositivos. La incorporación de tecnologías emergentes como la inteligencia artificial, el aprendizaje automático y el procesamiento de lenguaje natural en la arquitectura cliente-servidor en IoT podría mejorar aún más la capacidad de los dispositivos IoT para procesar y analizar datos de manera más inteligente y efectiva.

En resumen, la arquitectura cliente-servidor en IoT ha demostrado ser una herramienta efectiva para mejorar el procesamiento de datos en dispositivos IoT. Se espera que continúe creciendo en el futuro a medida que se desarrollen nuevas tecnologías y se integren en la vida cotidiana de las personas.

## 2.3. Recursos disponibles

Para nuestra implementación es necesario considerar las opciones de micro controladores disponibles para la simulación de nuestros auditores, tanto una implementación física como virtual formarán parte de nuestra propuesta y por lo tanto es necesario realizar la debida comparación entre componentes para este proyecto, centrandonos en Arduino y Raspberry Pi mostramos las características de los modelos en la siguiente tabla.

Bases	Arduino	Raspberry Pi
Licencia	Arduino es un proyecto de código abierto. Tanto su software como su diseño de hardware son de código abierto. Tanto el hardware como el software de Raspberry Pi son de código cerrado.	Tanto el hardware como el software son código cerrado.
Frecuencia de reloj	16 MHz (Arduino UNO)	Hasta 1.5GHz variando entre modelos
RAM	Requiere menos RAM (2kB)	Requiere mucho RAM (más de 1GB)
Consumo de energía	Consume aproximadamente 200 MW de energía.	Consume hasta 700 MW de energía.
Software	Programables usando C/C++.	Apoya su propio sistema operativo basado en Linux, y se puede instalar el SO que gustes.
Internet	Arduino no tiene apoyo a internet, se necesitan módulos externos.	Raspberry Pi tiene un puerto Ethernet y apoyo a WiFi.

Figura 3: Bases Arduino vs Raspberry Pi [5]

## 2.4. Generador de credenciales y su registro en un servidor AAA

OpenSSL y un servidor AAA (Autenticación, Autorización y Contabilidad) pueden trabajar juntos para crear un entorno seguro y confiable para dispositivos y usuarios de IoT. El proceso para generar credenciales que sirvan para autenticarse en el servidor AAA y establecer una conexión segura para la comunicación con el dispositivo IoT se describe a continuación:

1. El dispositivo IoT genera una clave pública y privada mediante OpenSSL. La clave privada se mantiene en secreto en el dispositivo, mientras que la clave pública se envía al servidor AAA.
2. El dispositivo IoT solicita un certificado digital del servidor AAA. La solicitud incluye la clave pública generada en el paso anterior y otra información sobre el dispositivo, como su nombre y dirección IP.
3. El servidor AAA verifica la identidad del dispositivo IoT al verificar la solicitud y la clave pública en su propia base de datos. Si el dispositivo está autorizado, el servidor genera un certificado digital que incluye la clave pública, lo firma con su propia clave privada y lo envía de regreso al dispositivo IoT.
4. El dispositivo IoT usa el certificado digital para establecer una conexión segura con el servidor AAA usando OpenSSL. El dispositivo envía su clave privada y el certificado digital al servidor, que verifica la autenticidad del certificado y establece una conexión segura para la comunicación. [6]

Otra opción que se puede considerar es un sistema 'certificateless', debido a que estos sistemas requieren de menor tiempo y poder de procesamiento de parte de nuestros artículos IoT. Los objetivos de este sistema son crear una autenticación de ambos lados, la anonimidad de la identidad, el centro de generación de llaves no conserva todas las llaves, la capacidad de actualizar tu llave, resistencia a ataques comunes, baja complejidad computacional y bajo costo de comunicación. Los algoritmo mas frecuentemente usado con 'certificateless' son bilinear pairing y ECC, debido a sus llaves cortas. Las principales ventajas de este sistema es que no necesita los gastos generales adicionales de guardar los certificados y provee flexibilidad y escalabilidad al sistema. Entre sus desventajas es que tienen una menor seguridad de los datos. [7] [8]

## 2.5. Protocolos posibles para IoT

Existen múltiples protocolos de comunicación para IoT, basado en algunas de sus características principales. Se realizó un análisis comparativo para determinar la compatibilidad y viabilidad de su uso en el proyecto a implementar de los dos mas comúnmente usados, MQTT y HTTP. [9]

Criterios	MQTT	HTTP
Arquitectura	Cliente/Bróker	Cliente/Servidor
Protocolo de transporte	TCP (MQTT-SN, puede usar UDP)	TCP
Seguridad	TLS/SSL	TLS/SSL
Puerto default	1883/8883 (TLS/SSL)	80/443 (TLS/SSL)
Encoding format	Binario	Binario
Modelo de licencia	Open Source	Libre

Cuadro 2: Comparativa de características generales de los protocolos de comunicación HTTP y MQTT [10]

En cuanto al uso de recursos de ambos protocolos podemos ver en las siguientes tablas una comparación del uso de batería así como de la velocidad de envío de las comunicaciones para ambos.

De acuerdo a la tabla 3 podemos ver que MQTT consume menos energía para cualquier periodo de tiempo probado. Esta diferencia es mas notoria en el caso de conexión a través de WiFi.



Tiempo (Segundos)	3G		WiFi	
	HTTP	MQTT	HTTP	MQTT
60	1.11553	0.72465	0.15839	0.01055
120	0.48697	0.32941	0.08774	0.00478
240	0.33277	0.16027	0.02897	0.00230
480	0.08263	0.07991	0.00824	0.00112

Cuadro 3: Comparativa de la batería usada para mantener la conexión. [11]

	3G		WiFi	
	HTTP	MQTT	HTTP	MQTT
%Batería/Hora	18.43 %	16.13 %	3.45 %	4.23 %
Mensajes/Hora	1708	160278	3628	263314
%Batería/Mensaje	0.01709	0.00010	0.00095	0.00002
Mensajes recibidos	240/1024	1024/1024	524/1024	1024/1024

Cuadro 4: Comparativa de ambos protocolos enviando 1024 mensajes.[11]

En cuanto a esta ultima tabla comparativa se puede decir que MQTT tiene un porcentaje de entrega del 100 % a pesar de consumir menos energía que HTTP.

Tomando en cuenta el análisis realizado y los datos para comparar ambos protocolos, concluimos que MQTT es una opción más viable para utilizar en un sistema de IoT, debido principalmente a su bajo uso de recursos, su rapidez y precisión en la comunicación.

## 2.6. Criptografía de clave pública

La Infraestructura de Clave Pública (PKI, por sus siglas en inglés) es un conjunto de componentes, políticas y procedimientos que se utilizan para crear, administrar, distribuir, almacenar y revocar certificados digitales y claves públicas. La PKI es esencial para la seguridad de las comunicaciones electrónicas y se utiliza en una variedad de aplicaciones, como la autenticación de usuarios, la firma digital de documentos, el cifrado de mensajes y la gestión de identidad.

La PKI se compone de varios elementos, como la Autoridad de Certificación (CA, por sus siglas en inglés), que es responsable de emitir los certificados digitales y gestionar la infraestructura de clave pública. También se incluyen los usuarios finales, que utilizan los certificados digitales para asegurar la identidad y la confidencialidad de las comunicaciones electrónicas, y los dispositivos de seguridad, como los tokens USB y las tarjetas inteligentes, que se utilizan para almacenar las claves privadas.

La PKI es una tecnología compleja que requiere una gestión adecuada para garantizar su eficacia y seguridad. Además, es importante que se establezcan políticas y procedimientos adecuados para la emisión, distribución y revocación de certificados digitales para evitar posibles vulnerabilidades en la infraestructura de clave pública.

En nuestra red proponemos la implementación de los siguientes componentes de una infraestructura de clave publica:

- Autoridad de certificación.
- Autoridad de registro.
- Autoridad de validación.
- Autoridad de repositorio.
- Software y políticas.

## 2.7. Bibliotecas disponibles para acelerar la implementación de los esquemas de clave pública

Para la implementación de un sistema de PKIs, se necesita la creación de un Certification Authority, que emite los certificados que confirman que los dispositivos son los que afirman ser; también hay que crear las claves públicas y privadas de cada dispositivo y una autoridad de validación que confirma que los certificados están vigentes.

La mayoría de los lenguajes de programación tienen librerías diseñadas para facilitar la implementación de PKIs. Algunos ejemplos de esto son la librería PKI en Java, las librerías CryptoSys PKI Pro y cryptography en Python; y la librería PKI en R, entre otros. [12] Estas librerías te ayudan a crear los hashes para la encriptación de los datos, generar strings de datos aleatorios, verificar firmas e incluso te permite generar los pares de llaves pública y privada para un dispositivo.

También se pueden utilizar librerías más especializadas para algunas de estas tareas, como hashlib para la aplicación de algoritmos de hash como SHA-2, SHA-3 y BLAKE2. También existe la librería certauth, que facilita la creación de CAs y la generación de certificados por parte de estos.

## 2.8. Virtualización de la red

La virtualización de la infraestructura de red se ha vuelto cada vez más popular debido a sus múltiples beneficios. Al virtualizar una infraestructura de red, se puede utilizar hardware físico existente de manera más eficiente, lo que permite a las empresas reducir el costo de adquisición de nuevo hardware. Además, permite una mayor flexibilidad en la implementación y gestión de la infraestructura de red, permitiendo a los administradores de TI cambiar rápidamente la configuración y agregar o eliminar servicios según sea necesario.

Otra ventaja clave de la virtualización es que puede mejorar la seguridad de la infraestructura de red. Esta permite a los administradores de TI aislar los servicios y aplicaciones de la red en entornos virtuales separados, lo que ayuda a prevenir la propagación de virus y otros malware en la red. Además, te permite implementar medidas de seguridad adicionales, como firewalls virtuales y sistemas de detección de intrusiones, para proteger la infraestructura de la red.

La virtualización también puede simplificar la gestión y el mantenimiento de la infraestructura de la red, ya que permite crear y administrar máquinas virtuales en un entorno centralizado. Esto puede mejorar la eficiencia operativa y reducir el tiempo de inactividad, ya que las máquinas virtuales se pueden mover fácilmente a otros servidores físicos en caso de falla de hardware.

Para implementar esta técnica es necesario lo siguiente:

- Selección del software de virtualización: Es necesario seleccionar una plataforma adecuada que pueda alojar la infraestructura de PKI. Puede utilizarse una herramienta de virtualización como VirtualBox, VMware, Hyper-V, entre otras.
- Creación de máquinas virtuales: Una vez que se tiene el software de virtualización, se pueden crear varias máquinas virtuales en el servidor físico para alojar los componentes de la infraestructura PKI. Es posible crear máquinas virtuales para el servidor de autenticación, servidor de clave pública, servidor de registro, cliente, entre otros.
- Instalación de componentes de la PKI: Una vez creadas las máquinas virtuales, se deben instalar los componentes necesarios de la infraestructura PKI. Esto puede incluir la instalación del servidor de autenticación, el servidor de registro y el servidor de clave pública.
- Configuración de los componentes de la PKI: Cada componente de la infraestructura PKI debe ser configurado. La configuración puede incluir la definición de políticas de seguridad, la configuración de certificados, la definición de roles y permisos, entre otros.

- Pruebas y validación: Una vez que se han instalado y configurado los componentes de la infraestructura PKI, se deben realizar pruebas y validaciones para asegurarse de que todo funcione correctamente. Esto puede incluir pruebas de autenticación, autorización y validación de certificados.

### 3. Algoritmos de encriptado posibles

#### 3.1. RSA

El algoritmo de encriptación RSA es un método de cifrado de clave pública que utiliza dos claves diferentes: una clave pública que se comparte ampliamente y una clave privada que se mantiene en secreto.

Para cifrar un mensaje con RSA, el remitente primero convierte el mensaje en un número entero y luego utiliza la clave pública del destinatario para cifrarlo. El destinatario, que es el único que tiene acceso a la clave privada correspondiente, puede descifrar el mensaje recibido.

La seguridad del algoritmo RSA se basa en la dificultad de factorizar grandes números enteros en números primos, que es el proceso inverso al utilizado para generar las claves. Debido a que factorizar números grandes es computacionalmente costoso, el algoritmo es muy seguro en la práctica, siempre y cuando se utilicen claves lo suficientemente largas.

#### 3.2. ECC

El algoritmo de encriptado ECC (Elliptic Curve Cryptography) es un método criptográfico que se basa en la utilización de curvas elípticas para generar claves públicas y privadas, que se utilizan para firmar y autenticar datos.

En este algoritmo, se utiliza una curva elíptica como un espacio de claves, donde cada punto en la curva representa una clave. El proceso de encriptación se lleva a cabo mediante la multiplicación de la clave pública por un valor aleatorio, mientras que la decodificación se realiza mediante la multiplicación de la clave privada por la clave cifrada.

ECC es un algoritmo de clave pública, lo que significa que se utilizan dos claves diferentes para cifrar y descifrar datos. La clave pública se comparte con otros usuarios, mientras que la clave privada se mantiene secreta. Este algoritmo es considerado más seguro que otros métodos criptográficos como RSA, ya que requiere claves más cortas y ofrece una mejor resistencia a los ataques basados en fuerza bruta.

#### 3.3. ECDSA

ECDSA (Elliptic Curve Digital Signature Algorithm) es un algoritmo de firma digital que utiliza curvas elípticas para generar claves públicas y privadas, y para verificar la autenticidad de los mensajes.

El proceso de firma digital con ECDSA comienza con la generación de una clave privada aleatoria y su correspondiente clave pública. Luego, el remitente de un mensaje utiliza su clave privada para generar una firma digital única para ese mensaje. La firma digital se envía junto con el mensaje al destinatario.

El destinatario utiliza la clave pública del remitente para verificar la autenticidad de la firma digital y, por lo tanto, la autenticidad del mensaje. Si la firma digital es auténtica, se considera que el mensaje es auténtico y que ha sido enviado por el remitente correspondiente.

ECDSA es considerado un algoritmo de firma digital seguro, ya que requiere claves más cortas que otros algoritmos de firma digital, como RSA, y ofrece una mejor resistencia a los ataques basados en fuerza bruta.

De acuerdo con Al-Zubaidie, 'Los algoritmos de criptografía de clave pública, en especial la criptografía de

curva elíptica (ECC) y el algoritmo de firma digital de curva elíptica (ECDSA), han sido objeto de atención por parte de muchos investigadores de diferentes instituciones debido a que estos algoritmos proporcionan seguridad y alto rendimiento cuando se utilizan en diversas áreas como la salud electrónica, banca electrónica, comercio electrónico, vehículos electrónicos y gobierno electrónico. [13].

#### 4. Implementaciones similares en la bibliografía

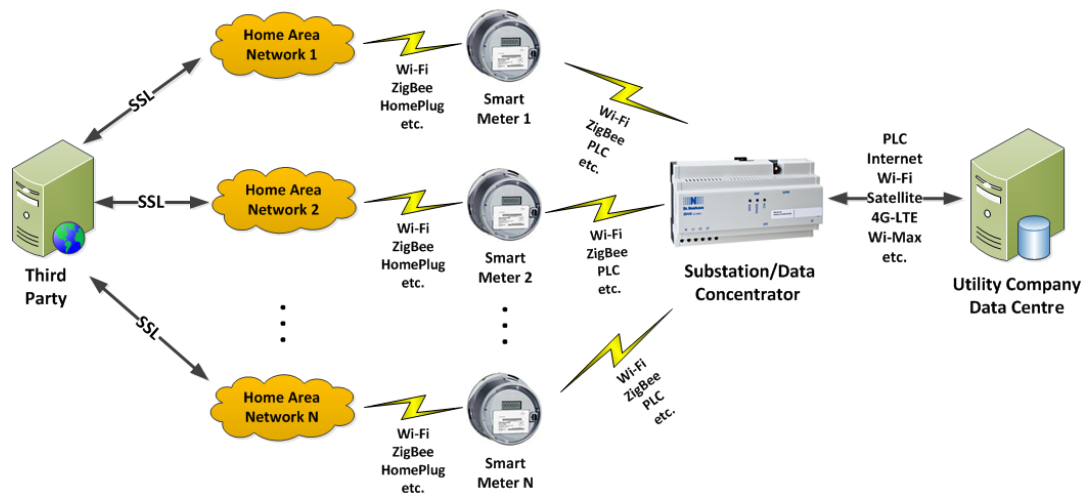


Figura 4: Arquitectura de red en una smart-grid [14]

El presente artículo [14] describe las principales funcionalidades de los componentes de un sistema de medición y control de una red inteligente (smart grid), que incluye la compañía de servicios públicos, la red de subestaciones y concentradores de datos, la red doméstica y el medidor inteligente. Cada componente tiene un papel específico en la gestión de datos de medición, facturación, monitoreo y control del consumo de energía en tiempo real; y la prestación de servicios de valor agregado para los usuarios finales.

Además, el artículo aborda la importancia de la seguridad en la red inteligente y cómo la certificación ECQV se utiliza para garantizar la autenticidad y la integridad de los datos de medición. Se menciona que, aunque el ZigBee SEP especifica que cada medidor inteligente debe estar equipado con un certificado ECQV, además el proceso de renovación y revocación de certificados no está definido en la especificación.

Finalmente, se discute la necesidad de encontrar un conjunto de algoritmos criptográficos que proporcionen la combinación adecuada de seguridad y facilidad de implementación para el sistema de medición y control de la red inteligente. El NIST sugiere AES, SHA-1 y RSA, y IEC 62351 especifica RSA-1024 para sistemas SCADA. El Standards Council of Canada y la Unión Europea definen requisitos de ciberseguridad para las redes inteligentes, pero no especifican un conjunto de algoritmos criptográficos para satisfacer los requisitos, excepto que el Standards Council of Canada especifica que se debe utilizar la función de hash segura SHA.

#### 5. Descripción y componentes de la propuesta

La arquitectura de red propuesta para el reto consistirá de dos micro-controladores Raspberry Pi que actuarán como nuestros auditores. Estos extraerán datos de manera periódica de la base de datos para simular el flujo de información de los sensores hacia los auditores, que ocurre cada 15 minutos de acuerdo a la base de datos. Los Raspberry Pis firman y encriptan los datos recibidos y los envían por medio de Internet a un bróker Mosquitto. El algoritmo de encriptado que vamos a usar es AES con ECDH para intercambio

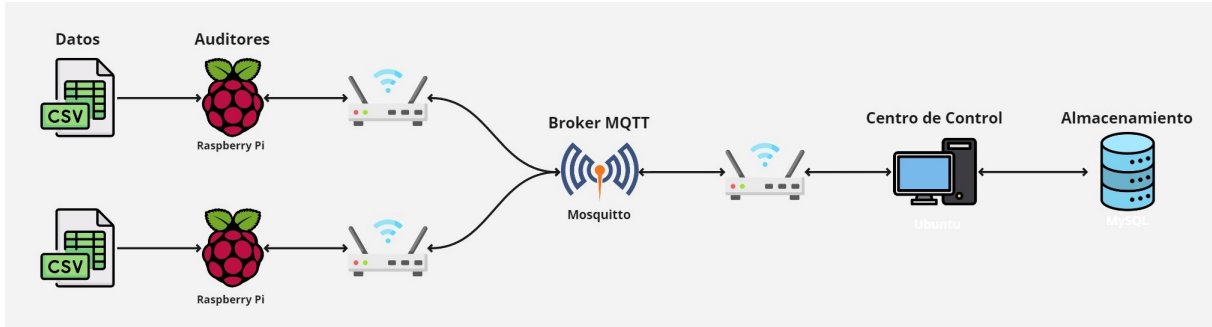


Figura 5: Arquitectura Propuesta

de claves y será acompañada con el algoritmo ECDSA para el firmado de los datos. Además se añadirán los certificados al paquete, estos serán usados para comprobar que los datos fueron enviados por un auditor autorizado. Adicionalmente, como una capa extra de seguridad, la conexión con el bróker utilizará TLS, que encriptará el tráfico en tránsito para mayor seguridad.

Posteriormente, los paquetes serán transmitidos por medio del bróker Mosquitto al Centro de Control. Los paquetes de datos serán descryptados al momento de llegar al Centro de Control, que además verificara la firma generada por los Raspberry Pi. El Centro de Control será además la Certification Authority, la cual estará encargada de generar y renovar los certificados usados por los auditores. Los datos serán almacenados en una base de datos relacional para su almacenamiento continuo. Los datos que almacenemos estarán encriptados at rest por medio del algoritmo AES con clave de 256 bits de longitud. Además la comunicación será bidireccional, es decir, el Centro de Control podrá comunicarse con los auditores, por ejemplo, enviándoles strings, que simularán las actualizaciones que ocurren en la vida real.

### 5.1. Seguridad física

Para mejorar la seguridad física de una red que utiliza smart-sensors, es importante tomar medidas para proteger los sensores mismos. Existen varias medidas que podemos implementar para garantizar la seguridad física de estos dispositivos.

En primer lugar, es fundamental que los sensores sean ubicados en una ubicación física segura, que no sea accesible fácilmente a personas no autorizadas. Esto se puede lograr mediante el uso de gabinetes cerrados con llave, jaulas, o habitaciones con acceso restringido.

En segundo lugar, es recomendable utilizar sensores a prueba de manipulaciones que puedan detectar si alguien intenta abrirlos o manipularlos. Estos sensores pueden enviar alertas al personal de seguridad si una persona no autorizada intenta acceder a ellos.

En tercer lugar, es importante utilizar medidas de autenticación sólidas para garantizar que solo los usuarios autorizados puedan acceder a los datos de los sensores. Esto puede incluir el uso de contraseñas fuertes, autenticación de dos factores o biométrica.

En cuarto lugar, se deben realizar chequeos regulares de mantenimiento en los sensores para garantizar que estén funcionando correctamente y que no hayan sido manipulados. Esto puede incluir verificar la integridad de los sensores, reemplazar las baterías y actualizar el firmware.

Por último, se debe utilizar la supervisión física, como cámaras de seguridad o patrullas de seguridad, para garantizar que la ubicación de los sensores no sea comprometida.

## 5.2. Certificados

Un certificado SSL (Secure Sockets Layer) no encripta directamente los datos. En cambio, lo que hace es garantizar la autenticidad y la integridad de la conexión. De forma compacta nuestra propuesta funciona de la siguiente forma:

1. Certificados autofirmados: Estos certificados son generados y firmados por el propio usuario, en lugar de una autoridad de certificación (AC) confiable, esto significa que crearemos nuestra propia CA.
2. Abrimos una terminal o símbolo del sistema.
3. Ejecutamos el siguiente comando OpenSSL para generar una nueva clave privada y un certificado autofirmado: `openssl x509 -req -in ca.csr -signkey broker.key -out ca.crt -days 365`
4. Este comando creará una nueva clave privada `ec` de 256 bits (`ca.key`) y un certificado autofirmado `X,509` (`broker.key`) válido por 365 días.
5. Se nos solicitará ingresar una frase de contraseña para la clave privada.
6. Elegimos una contraseña segura que necesitaremos para desbloquear la clave privada cuando la usemos en nuestra aplicación.
7. A continuación, se nos pedirá proporcionar cierta información para el sujeto del certificado, como el país, estado, organización y nombre de dominio (Common Name). Completamos estos detalles requeridos.
8. Una vez que hayamos completado el proceso, tendremos un certificado SSL autofirmado para el servidor, la CA y los clientes (`componente.crt`) y una clave privada (`ca.key`). Podemos utilizar estos archivos en nuestro entorno de desarrollo o pruebas.

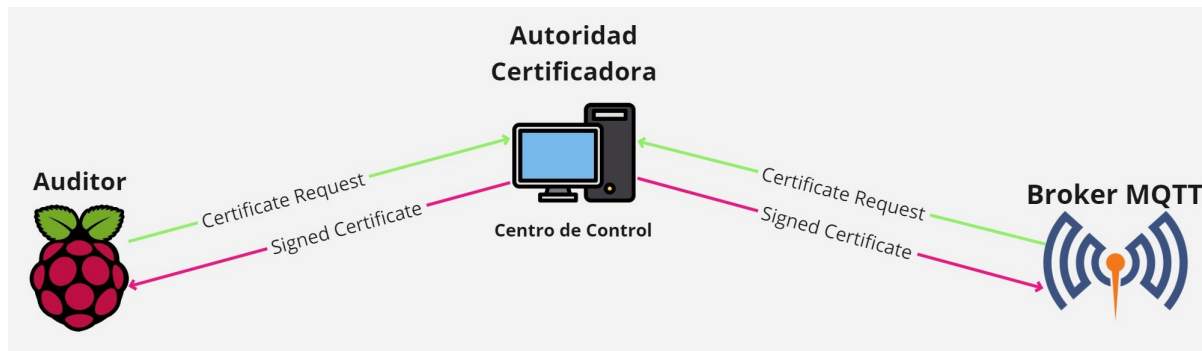


Figura 6: Solicitud y firma de certificados en nuestra red

### 5.3. Encriptado de los datos

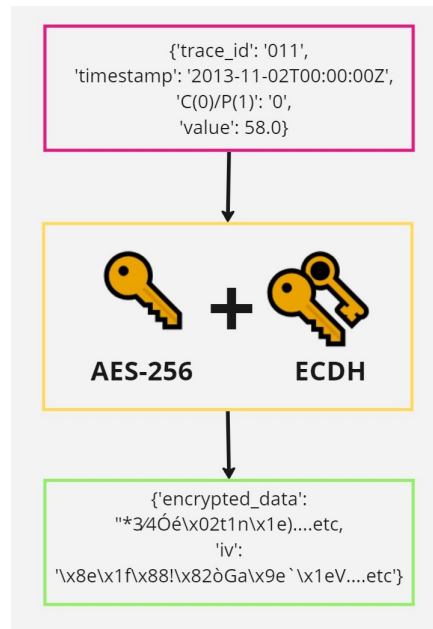


Figura 7: Encriptado (AES) y canal seguro (ECDH)

Para garantizar la seguridad de los datos durante su tránsito desde el auditor hasta el centro de control a través de Internet, hemos decidido utilizar el algoritmo AES-256 (Advanced Encryption Standard) para encriptar los datos. AES es un algoritmo de encriptación simétrica que ofrece una mayor eficiencia en términos de procesamiento y tamaño de clave en comparación con RSA u otros algoritmos asimétricos. Para acordar la clave secreta necesaria para la encriptación con AES, utilizaremos ECDH como algoritmo para la generación e intercambio de la clave pública. Esto nos permitirá compartir de forma segura la clave secreta entre el auditor y el centro de control, mientras que la encriptación y desencriptación de los datos en sí se realizará utilizando AES-256.

Esta combinación de ECDH para compartir la clave secreta y AES-256 para encriptar los datos nos brindará un equilibrio entre seguridad end-to-end y eficiencia en nuestra solución. De forma resumida el proceso es el siguiente:

1. Generación de claves ECDH: Tanto el auditor como el centro de control generan su par de claves ECDH, que consta de una clave pública y una clave privada.
2. Intercambio de claves públicas: El auditor y el centro de control intercambian sus claves públicas ECDH de manera segura para establecer una comunicación segura y confiable.
3. Generación de clave secreta: El auditor genera una clave secreta aleatoria para la encriptación con AES-256. Esta clave se mantendrá en secreto y solo se compartirá con el centro de control de forma segura utilizando ECDH.
4. Encriptación de la clave secreta: El auditor encripta la clave secreta generada utilizando la clave pública ECDH del centro de control. Esto garantiza que solo el centro de control, con su clave privada correspondiente, pueda desencriptar la clave secreta.
5. Envío seguro de la clave encriptada: El auditor envía la clave secreta encriptada al centro de control a través de un canal seguro. Esto puede incluir métodos como el uso de TLS para proteger la comunicación.

6. Descriptación de la clave secreta: El centro de control utiliza su clave privada ECDH para descriptar la clave secreta recibida, lo que le permite obtener la clave original generada por el auditor.
7. Encriptación de los datos con AES-256: Utilizando la clave secreta compartida, el auditor encripta los datos utilizando el algoritmo AES-256. Esto asegura que los datos estén protegidos y solo puedan ser descriptados por el centro de control utilizando la misma clave secreta.
8. Envío de los datos encriptados: El auditor envía los datos encriptados al centro de control a través de Internet, utilizando protocolos seguros como TLS para garantizar la confidencialidad y la integridad de la transmisión.
9. Descriptación de los datos en el centro de control: Utilizando la misma clave secreta compartida, el centro de control descripta los datos recibidos, lo que le permite acceder a los datos originales en su forma legible.
10. Este proceso se repetirá cada 8 horas para asegurar de que incluso si esta llave fuese comprometida, el daño sea minimizado,

## 5.4. Firma de los datos

Para realizar el firmado de los datos, elegimos el algoritmo de firmado con curvas elípticas para balancear el bajo poder computacional disponible en nuestros auditores, por lo tanto seleccionamos el método eficiente ECDSA. El proceso que sigue nuestro algoritmo es el siguiente de forma resumida:

1. Generación del valor  $K$  determinístico: Se utiliza la función `deterministic_generate_k` para generar el valor  $K$  de manera determinística. Esta función sigue los pasos del algoritmo de generación de  $K$  determinístico de ECDSA. Se prepara la clave y el mensaje, se generan valores intermedios  $V$  y  $K$  utilizando la función HMAC con el algoritmo de hash SHA256, y finalmente se convierte el valor  $K$  a un entero.
2. Firma ECDSA: Para firmar un mensaje, se sigue el proceso de firma ECDSA. Se calcula el hash del mensaje utilizando el algoritmo SHA256. Luego, se genera el valor  $K$  determinístico utilizando la función `deterministic_generate_k`.
3. A continuación, se calcula la clave pública correspondiente a la clave privada utilizando la multiplicación de curvas elípticas. Se calcula el punto aleatorio de firma utilizando la multiplicación de curvas elípticas con el valor  $K$ .
4. Se calcula el valor  $r$  como el resultado de tomar el módulo  $N$  del componente  $x$  del punto aleatorio de firma. Finalmente, se calcula el valor  $s$  utilizando fórmulas y operaciones matemáticas basadas en los componentes del mensaje, la clave privada, el valor  $r$  y el valor  $K$ .
5. Verificación de firma ECDSA: Para verificar una firma ECDSA, se sigue el proceso de verificación ECDSA. Se calcula el hash del mensaje utilizando el algoritmo SHA256. Luego, se calcula el valor  $w$  como el inverso multiplicativo de  $s$  módulo  $N$ .
6. Se calculan los puntos  $u_1$  y  $u_2$  utilizando multiplicaciones de curvas elípticas con los componentes del mensaje, el valor  $w$  y el valor  $r$ .
7. A continuación, se calcula el punto de verificación sumando los puntos  $u_1$  y  $u_2$  utilizando la operación `ECadd`. Finalmente, se verifica si el valor  $r$  de la firma coincide con el componente  $x$  del punto de verificación.



## 5.5. Bróker Mosquitto

MQTT opera con el patrón publish-subscribe, que separa al remitente y al receptor del mensaje, lo que permite una comunicación eficiente entre los dispositivos IoT. La confiabilidad de MQTT se atribuye a su uso de un bróker, este funciona como un servidor que recibe mensajes de los publishers y los enruta a los subscribers apropiados según el tópico del mensaje.

Mosquitto es una implementación de código abierto del protocolo MQTT que ofrece numerosas funciones y ventajas para proyectos de IoT. Algunas de las razones por las que se ha elegido Mosquitto son:

- **Licencia y flexibilidad:** Se publica bajo la licencia Eclipse Public License, lo que brinda flexibilidad y la posibilidad de personalizar el bróker según los requisitos específicos del proyecto.
- **Seguridad:** Ofrece soporte para certificados autofirmados mediante el uso de ECDSA (Elliptic Curve Digital Signature Algorithm), lo que mejora la seguridad al permitir el cifrado y la autenticación de las conexiones MQTT.
- **Rendimiento y escalabilidad:** Es capaz de manejar millones de conexiones MQTT y procesar una gran cantidad de mensajes por segundo, lo que garantiza un alto rendimiento y escalabilidad.
- **Baja latencia:** Garantiza una entrega rápida y eficiente de mensajes, lo que es crucial en entornos de IoT donde la latencia puede ser crítica.
- **Mecanismos de autenticación:** Ofrece la posibilidad de integrar una gestión de usuarios mediante bases de datos y listas de control de acceso, lo que permite un control de acceso granular y una autenticación segura.

La naturaleza de código abierto de Mosquitto fomenta las contribuciones de la comunidad y las mejoras continuas, lo que garantiza que el bróker se mantenga actualizado con las necesidades cambiantes de los proyectos de IoT.

En resumen, Mosquitto es una elección adecuada como bróker MQTT debido a su confiabilidad, escalabilidad, rendimiento, funciones de seguridad y naturaleza de código abierto. Al utilizar Mosquitto con certificados autofirmados mediante ECDSA, el proyecto puede lograr una comunicación segura y confiable dentro del ecosistema de IoT, abordando los requisitos específicos del contexto de IoT. Cabe destacar, que el broker proporciona la funcionalidad de enrutar mensajes y asegurar la comunicación cifrada mediante TLS. Sin embargo, el centro de control, actuando como una autoridad de certificación (CA), emite los certificados utilizados para autenticar y cifrar las conexiones MQTT entre los auditores, el broker de Mosquitto y el mismo centro de control. El centro de control tiene el control en términos de seguridad al emitir y gestionar los certificados utilizados en el ecosistema de comunicación MQTT.

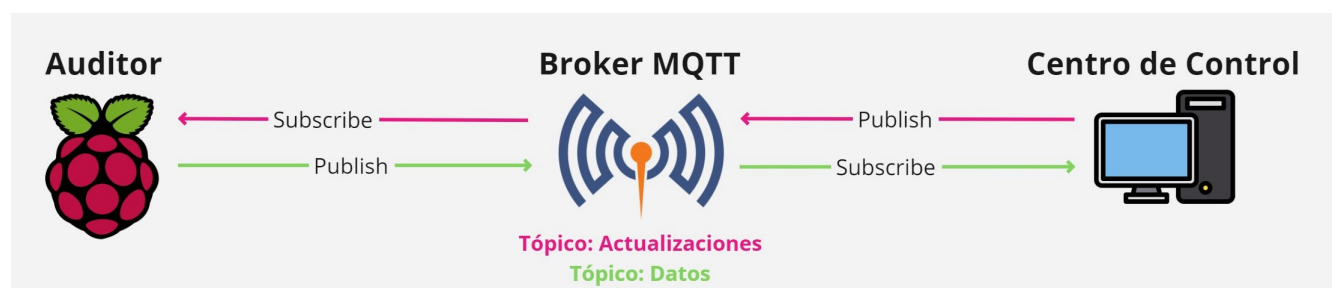


Figura 8: Comunicación bidireccional usando Mosquitto como bróker

## 5.6. Almacenamiento de los datos

Para el almacenamiento de los datos recomendamos usar una base de datos relacional montada en un servidor NAS o similar. Utilizaremos MySQL, que se encargara del manejo y encriptado de los datos, además de la autorización y autenticación de los usuarios. Los datos en la base de datos serán encriptados por el algoritmo AES-256, que es el estándar de la industria y recomendado por el NIST. Para el acceso de la base de datos se generan usuarios y contraseñas que son verificados con los certificados propios de MySQL.

Esta implementación podría ser expandida en un futuro con el montaje de la base de datos con un servidor de la nube basado en servicios especializados en esto. Esto no modificaría significativamente la arquitectura y nos traería ventajas como mayor libertad de accesos y mayor almacenamiento.

### 5.6.1. Costos de implementación aproximados

Producto / Servicio	Componentes	Precio aproximado
Audidores (2)	Raspberry Pi 4 4gb	~100 USD * 2 [15]
Módem (2)	HUAWEI Router WIFI 6 AX2 1500Mbps White	~45 USD * 2 [16].

Cuadro 5: Tabla de costos aproximados consultados el 04/17/2023

Precios de almacenamiento	USD por GB
Primeros 50 TB/mes	\$0.023
Siguientes 450 TB/mes	\$0.022
Más de 500 TB/mes	\$0.021

Cuadro 6: Tabla de costos según la página oficial de Amazon S3 [17]

## 6. Metodología y desarrollo de la solución seleccionada

La primera etapa consiste en el arranque, donde encendemos y configuramos los audidores y el centro de control. Para fines prácticos, utilizaremos dos máquinas virtuales con Ubuntu.

Antes de proceder, es importante establecer correctamente cada elemento de la red. Definiremos los archivos que debe tener cada componente:

1. Servidor MQTT: Utilizaremos el bróker Mosquitto en una maquina virtual con Ubuntu. Los archivos necesarios son los siguientes:
  - ca.crt
  - server.crt
  - server.key
  - client.crt
  - client.key
2. Auditor 1: Los archivos requeridos para el primer auditor son:
  - client1.crt
  - client1.key

- Subscriber.py
- Publisher.py

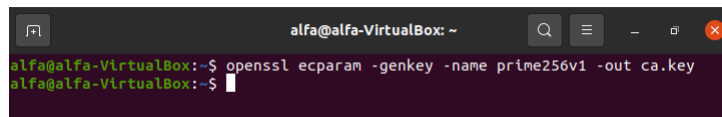
3. Auditor 2: Los archivos requeridos para el segundo auditor son:

- client2.crt
- client2.key
- Subscriber.py
- Publisher.py

## 6.1. Creación y firma de certificados

### 6.1.1. Certificados Centro de Control

Primero generaremos la clave y certificados de la autoridad certificadora (CA) que en nuestra propuesta sería el Centro de Control:



```
alfa@alfa-VirtualBox: ~  
alfa@alfa-VirtualBox:~$ openssl ecparam -genkey -name prime256v1 -out ca.key  
alfa@alfa-VirtualBox:~$
```

Figura 9: Llave privada de la CA

Luego generamos el certificate request (CSR) de la CA para autofirmarlo con la clave generada en el punto anterior (es importante que en Common Name se encuentre el nombre o dirección de nuestro bróker MQTT):

Ahora ya firmamos el certificado con nuestra llave generada anteriormente, esto para tener nuestro certificado Raíz con el cual posteriormente se podrán verificar los demás certificados emitidos por la CA (antes de firmar los certificados es necesario cerciorarse de que vienen de quien dice ser y que son validos para aplicarlos en nuestra infraestructura de red):

Después de haber creado a nuestra autoridad certificadora vamos a crear y firmar los certificados de los demás componentes de nuestra red, para empezar vamos a comenzar con el del bróker.

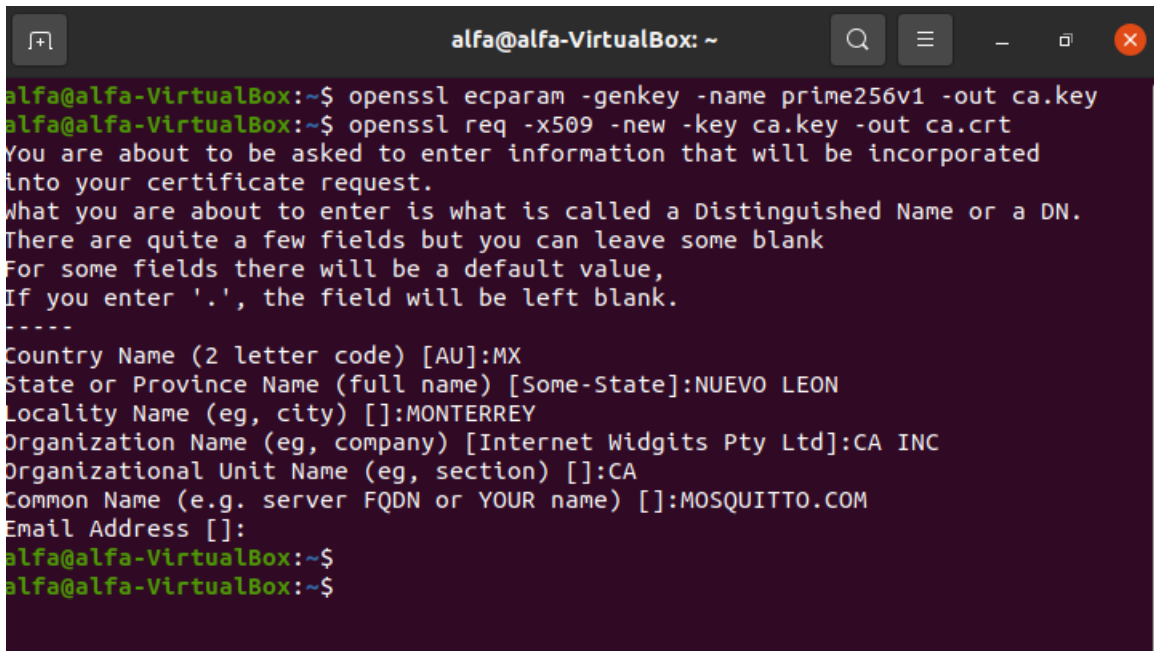
### 6.1.2. Certificados bróker

Creamos una solicitud de firma de certificado utilizando la clave del servidor para enviar a la CA para la verificación de identidad:

- Asignamos a la organización un nombre como "Localhost MQTT Broker Inc". y el nombre común debe ser localhost o el dominio exacto que usa para conectarse al bróker mqtt.

Ahora, actuando como la CA, recibimos la solicitud del servidor para su firma. Hemos verificado que el servidor es quien dice ser (un bróker MQTT que opera en la dirección del bróker), así que creamos un nuevo certificado y firmamos con todo el poder de la autoridad certificadora.

Con esto ya tenemos nuestro certificado firmado y validado por la CA que vamos poder utilizar para establecer la conexión entre el bróker y el centro de control.

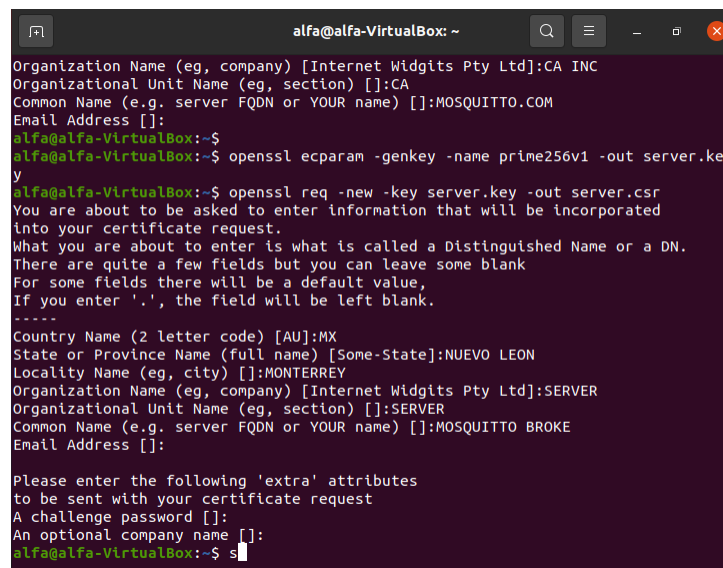


```

alfa@alfa-VirtualBox: ~
alfa@alfa-VirtualBox:~$ openssl ecparam -genkey -name prime256v1 -out ca.key
alfa@alfa-VirtualBox:~$ openssl req -x509 -new -key ca.key -out ca.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:MX
State or Province Name (full name) [Some-State]:NUEVO LEON
Locality Name (eg, city) []:MONTERREY
Organization Name (eg, company) [Internet Widgits Pty Ltd]:CA INC
Organizational Unit Name (eg, section) []:CA
Common Name (e.g. server FQDN or YOUR name) []:MOSQUITTO.COM
Email Address []:
alfa@alfa-VirtualBox:~$
alfa@alfa-VirtualBox:~$

```

Figura 10: Creación y firmado de la CA (obtenemos el certificado raíz)



```

Organization Name (eg, company) [Internet Widgits Pty Ltd]:CA INC
Organizational Unit Name (eg, section) []:CA
Common Name (e.g. server FQDN or YOUR name) []:MOSQUITTO.COM
Email Address []:
alfa@alfa-VirtualBox:~$
alfa@alfa-VirtualBox:~$ openssl ecparam -genkey -name prime256v1 -out server.key
alfa@alfa-VirtualBox:~$ openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:MX
State or Province Name (full name) [Some-State]:NUEVO LEON
Locality Name (eg, city) []:MONTERREY
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SERVER
Organizational Unit Name (eg, section) []:SERVER
Common Name (e.g. server FQDN or YOUR name) []:MOSQUITTO BROKE
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
alfa@alfa-VirtualBox:~$ s

```

Figura 11: Creamos el CRS del bróker

11

### 6.1.3. Certificados auditores

A continuación mostraremos el proceso para un auditor, tomando en cuenta que este sera esencialmente el mismo para cualquier otro componente o auditor que se desee agregar y autenticar:

Creamos una solicitud de firma de certificado (CSR) para el auditor:

```

alfa@alfa-VirtualBox:~$ openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.k
ey -CAcreateserial -out server.crt
Signature ok
subject=C = MX, ST = NUEVO LEON, L = MONTERREY, O = SERVER, OU = SERVER, CN = M
OSQUITTO BROKE
Getting CA Private Key
alfa@alfa-VirtualBox:~$ s

```

Figura 12: Firmado del CRS del bróker con la llave de la CA  
12

```

alfa@alfa-VirtualBox:~$ openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.k
ey -CAcreateserial -out server.crt
Signature ok
subject=C = MX, ST = NUEVO LEON, L = MONTERREY, O = SERVER, OU = SERVER, CN = M
OSQUITTO BROKE
Getting CA Private Key

```

Figura 13: Firmado del CSR del bróker con la llave de la CA

Este es un ejemplo de como se ve un CSR que posteriormente tiene que ser firmado por la CA:

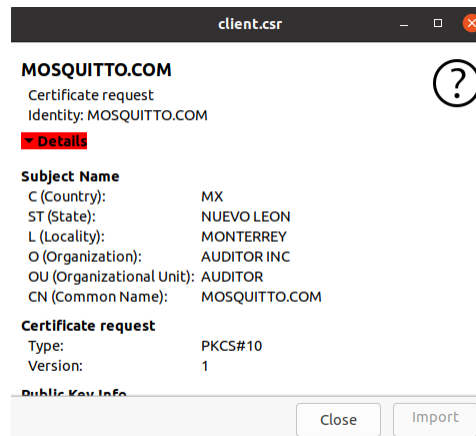


Figura 14: Ejemplo del certificate Request de un auditor

Firmamos el CSR del auditor con la CA para crear el certificado del auditor (esto lo hacemos en el Centro de Control):

```

alfa@alfa-VirtualBox:~$ openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.k
ey -CAcreateserial -out client.crt
Signature ok
subject=C = MX, ST = NUEVO LEON, L = MONTERREY, O = AUDITOR INC, OU = AUDITOR,
CN = MOSQUITTO.COM
Getting CA Private Key
alfa@alfa-VirtualBox:~$

```

Figura 15: Ejemplo del certificado firmado de un auditor

## 6.2. Instalar y configurar Mosquitto

Primero instalamos Mosquitto y Mosquitto clients utilizando sudo

Luego vamos a correr el bróker para ver si todo está configurado correctamente, en la figura 17 podemos ver que todo está en orden y el bróker está corriendo, además podemos ver que por defecto usa el puerto 1883, cosa que tendremos que cambiar para que use el puerto 8883 y así poder implementar TLS:

En esta parte tenemos que entrar al archivo de configuración del bróker y cambiar los parámetros para

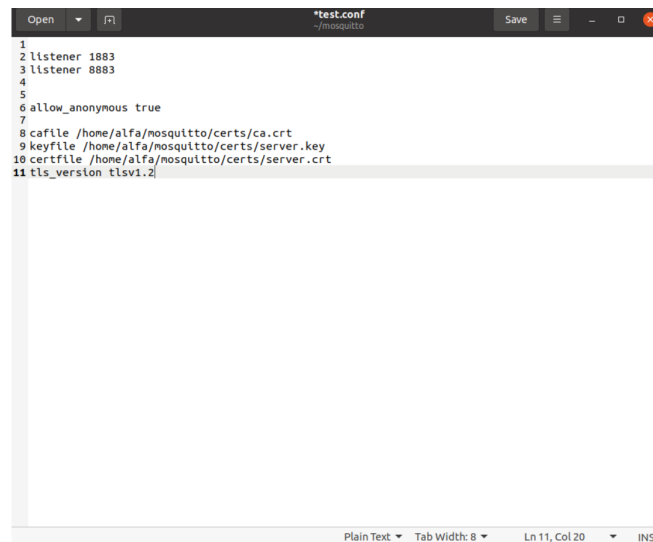
```
alfa@alfa-VirtualBox:~$ sudo apt-get install mosquitto-clients
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  mosquitto-clients
0 upgraded, 1 newly installed, 0 to remove and 365 not upgraded.
Need to get 123 kB of archives.
After this operation, 334 kB of additional disk space will be used.
Get:1 http://ppa.launchpad.net/mosquitto-dev/mosquitto-ppa/ubuntu focal/main amd64 mosquitto-clients amd64 2.0.15a-0mosquitto1-focal1 [123 kB]
Fetched 123 kB in 1s (137 kB/s)
Selecting previously unselected package mosquitto-clients.
(Reading database ... 158108 files and directories currently installed.)
Preparing to unpack ../mosquitto-clients_2.0.15a-0mosquitto1-focal1_amd64.deb
```

Figura 16: Instalación de Mosquitto

```
alfa@alfa-VirtualBox:~$ systemctl stop mosquitto.service
alfa@alfa-VirtualBox:~$ mosquitto -v
1686207981: mosquitto version 2.0.15 starting
1686207981: Using default config.
1686207981: Starting in local only mode. Connections will only be possible from
clients running on this machine.
1686207981: Create a configuration file which defines a listener to allow remot
e access.
1686207981: For more details see https://mosquitto.org/documentation/authentica
tion-methods/
1686207981: Opening ipv4 listen socket on port 1883.
1686207981: Opening ipv6 listen socket on port 1883.
1686207981: mosquitto version 2.0.15 running
```

Figura 17: Como se ve el bróker corriendo en la maquina virtual

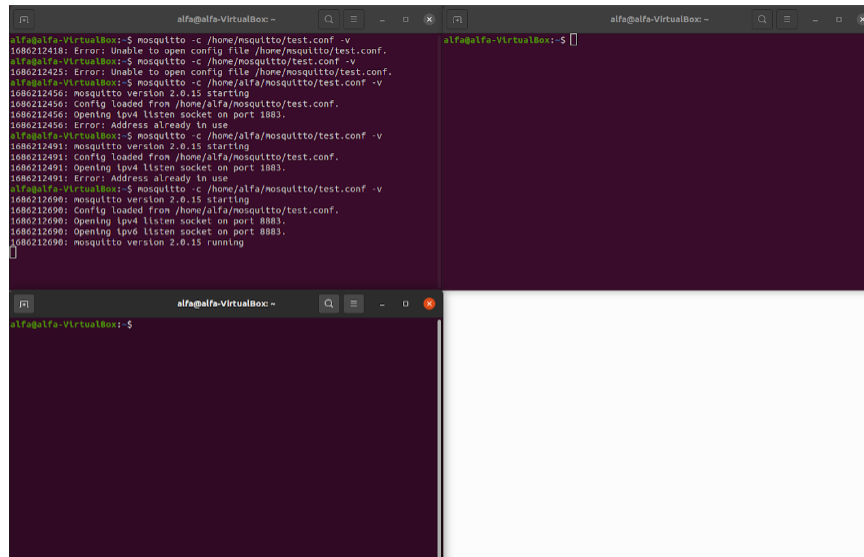
que se use el puerto 8883 y cargar los certificados que vamos a utilizar, además de que momentaneamente permitimos que cualquier IP pueda establecer comunicación con el bróker, cosa que después debemos de cambiar cuando contemos con equipos bien definidos para agregar una capa extra de seguridad:



```
*test.conf
~/mosquitto
1
2 listener 1883
3 listener 8883
4
5
6 allow_anonymous true
7
8 cafile /home/alfa/mosquitto/certs/ca.crt
9 keyfile /home/alfa/mosquitto/certs/server.key
10 certfile /home/alfa/mosquitto/certs/server.crt
11 tls_version tlsv1.2
```

Figura 18: Ejemplo de archivo de configuración de nuestro bróker

Ahora vamos a correr el bróker de Mosquitto usando el nuevo archivo de configuración que creamos anteriormente, en la figura 19 podemos observar que ahora efectivamente se está usando el puerto 8883. Por lo tanto, podremos usar TLS con los certificados que creamos anteriormente:



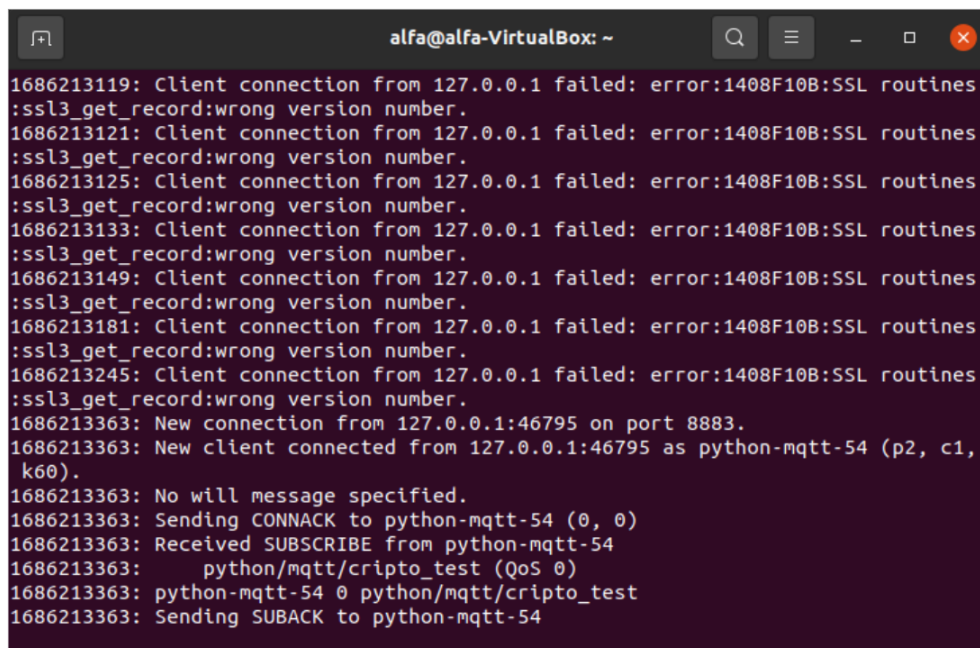
```

alfa@alfa-VirtualBox: ~$ mosquitto -c /home/mosquitto/test.conf -v
1686212418: Error: Unable to open config file /home/mosquitto/test.conf.
alfa@alfa-VirtualBox: ~$ mosquitto -c /home/mosquitto/test.conf -v
1686212425: Error: Unable to open config file /home/mosquitto/test.conf.
alfa@alfa-VirtualBox: ~$ mosquitto -c /home/alfa/mosquitto/test.conf -v
1686212456: mosquitto version 2.0.15 starting
1686212456: Config loaded from /home/alfa/mosquitto/test.conf.
1686212456: Opening ipv4 listen socket on port 1883.
1686212456: Error: Address already in use
alfa@alfa-VirtualBox: ~$ mosquitto -c /home/alfa/mosquitto/test.conf -v
1686212491: mosquitto version 2.0.15 starting
1686212491: Config loaded from /home/alfa/mosquitto/test.conf.
1686212491: Opening ipv4 listen socket on port 1883.
1686212491: Error: Address already in use
alfa@alfa-VirtualBox: ~$ mosquitto -c /home/alfa/mosquitto/test.conf -v
1686212690: mosquitto version 2.0.15 starting
1686212690: Config loaded from /home/alfa/mosquitto/test.conf.
1686212690: Opening ipv4 listen socket on port 8883.
1686212690: Opening ipv4 listen socket on port 8883.
1686212690: mosquitto version 2.0.15 running

```

Figura 19: Ejemplo de archivo de configuración de nuestro bróker

Ahora en la figura 20 podemos ver como se ve una conexión desde el bróker, en la parte de arriba podemos ver como se ven conexiones fallidas hechas desde el auditor al bróker y en la parte de abajo podemos ver como se conecta exitosamente y ahora se puede comenzar a publicar y suscribirse a un tópico en específico:



```

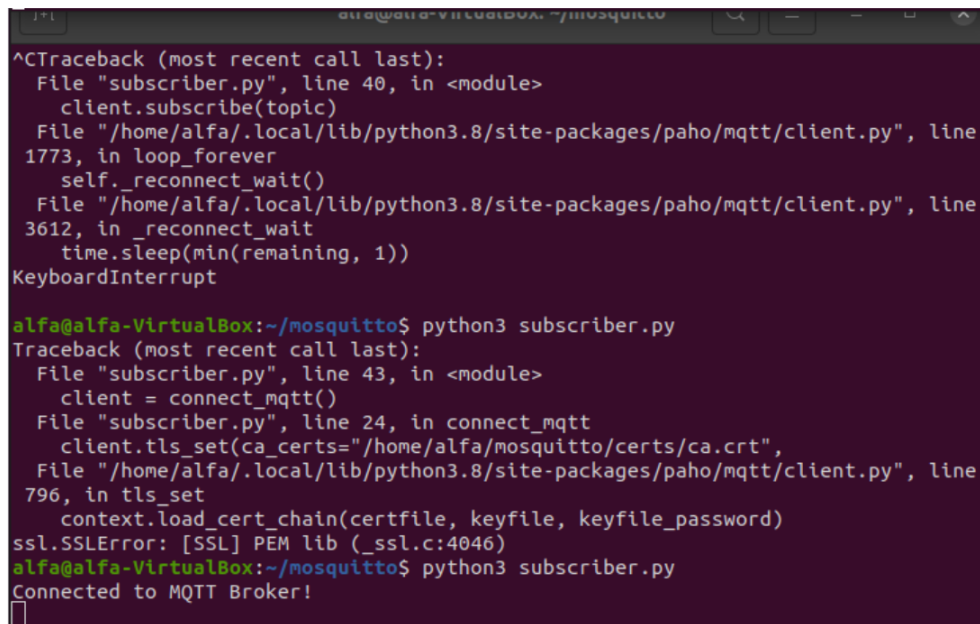
alfa@alfa-VirtualBox: ~$ mosquitto -c /home/alfa/mosquitto/test.conf -v
1686213119: Client connection from 127.0.0.1 failed: error:1408F10B:SSL routines
:ssl3_get_record:wrong version number.
1686213121: Client connection from 127.0.0.1 failed: error:1408F10B:SSL routines
:ssl3_get_record:wrong version number.
1686213125: Client connection from 127.0.0.1 failed: error:1408F10B:SSL routines
:ssl3_get_record:wrong version number.
1686213133: Client connection from 127.0.0.1 failed: error:1408F10B:SSL routines
:ssl3_get_record:wrong version number.
1686213149: Client connection from 127.0.0.1 failed: error:1408F10B:SSL routines
:ssl3_get_record:wrong version number.
1686213181: Client connection from 127.0.0.1 failed: error:1408F10B:SSL routines
:ssl3_get_record:wrong version number.
1686213245: Client connection from 127.0.0.1 failed: error:1408F10B:SSL routines
:ssl3_get_record:wrong version number.
1686213363: New connection from 127.0.0.1:46795 on port 8883.
1686213363: New client connected from 127.0.0.1:46795 as python-mqtt-54 (p2, c1,
k60).
1686213363: No will message specified.
1686213363: Sending CONNACK to python-mqtt-54 (0, 0)
1686213363: Received SUBSCRIBE from python-mqtt-54
1686213363: python/mqtt/cripto_test (QoS 0)
1686213363: python-mqtt-54 0 python/mqtt/cripto_test
1686213363: Sending SUBACK to python-mqtt-54

```

Figura 20: Conexiones hechas al bróker usando el nuevo archivo de configuración

En la figura 21 podemos ver como nos suscribimos al bróker usando el código subscriber.py, además de ver como falla la conexión al intentar validarnos con certificados diferentes a los que creamos al inicio. En la parte de abajo ya con el uso de los certificados verdaderos podemos ver que se logra la conexión con el bróker:



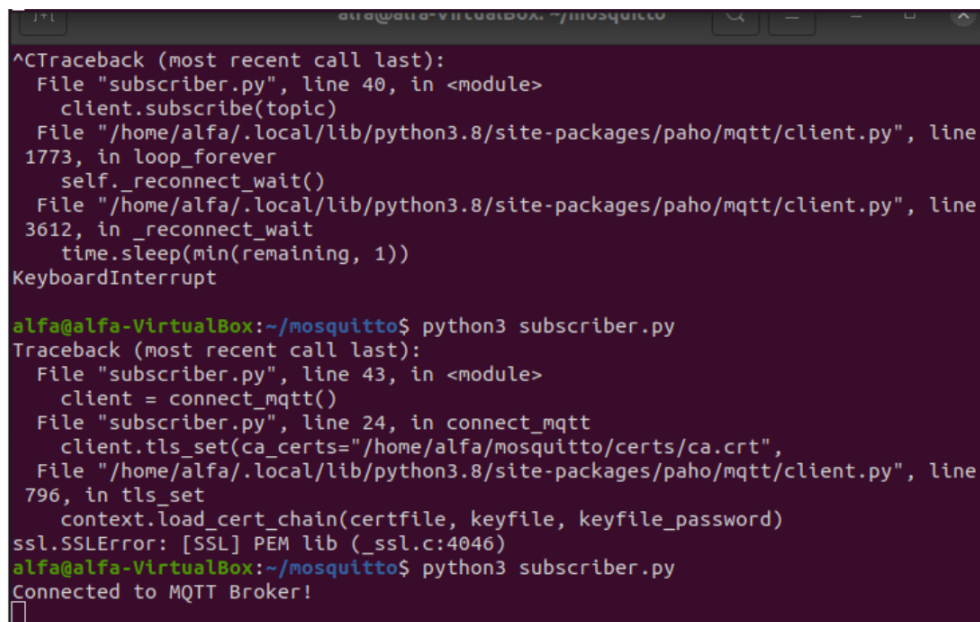
A terminal window with a dark background and light-colored text. It shows a Python script named 'subscriber.py' being executed. The script attempts to connect to an MQTT broker using TLS certificates. The connection fails with an 'ssl.SSLError: [SSL] PEM lib (\_ssl.c:4046)' error. The terminal output includes a traceback showing the error occurred in the 'connect\_mqtt' function of 'paho/mqtt/client.py'. The user then runs the script again, and it successfully connects to the MQTT broker, displaying 'Connected to MQTT Broker!' at the end.

```
^CTraceback (most recent call last):
  File "subscriber.py", line 40, in <module>
    client.subscribe(topic)
  File "/home/alfa/.local/lib/python3.8/site-packages/paho/mqtt/client.py", line
1773, in loop_forever
    self._reconnect_wait()
  File "/home/alfa/.local/lib/python3.8/site-packages/paho/mqtt/client.py", line
3612, in _reconnect_wait
    time.sleep(min(remaining, 1))
KeyboardInterrupt

alfa@alfa-VirtualBox:~/mosquitto$ python3 subscriber.py
Traceback (most recent call last):
  File "subscriber.py", line 43, in <module>
    client = connect_mqtt()
  File "subscriber.py", line 24, in connect_mqtt
    client.tls_set(ca_certs="/home/alfa/mosquitto/certs/ca.crt",
  File "/home/alfa/.local/lib/python3.8/site-packages/paho/mqtt/client.py", line
796, in tls_set
    context.load_cert_chain(certfile, keyfile, keyfile_password)
ssl.SSLError: [SSL] PEM lib (_ssl.c:4046)
alfa@alfa-VirtualBox:~/mosquitto$ python3 subscriber.py
Connected to MQTT Broker!
```

Figura 21: Ejemplo de conexión fallida con certificados alterados

En la figura 22 podemos ver como funciona el bróker con las 2 partes principales, el suscriptor y el publicador, además de que podemos ver que las trazas se publican y se reciben de forma correcta y completa.

A terminal window similar to the one in Figure 21, but showing a successful connection. The first attempt fails with the same SSL error. The user then runs the script again, and it successfully connects to the MQTT broker, displaying 'Connected to MQTT Broker!' at the end.

```
^CTraceback (most recent call last):
  File "subscriber.py", line 40, in <module>
    client.subscribe(topic)
  File "/home/alfa/.local/lib/python3.8/site-packages/paho/mqtt/client.py", line
1773, in loop_forever
    self._reconnect_wait()
  File "/home/alfa/.local/lib/python3.8/site-packages/paho/mqtt/client.py", line
3612, in _reconnect_wait
    time.sleep(min(remaining, 1))
KeyboardInterrupt

alfa@alfa-VirtualBox:~/mosquitto$ python3 subscriber.py
Traceback (most recent call last):
  File "subscriber.py", line 43, in <module>
    client = connect_mqtt()
  File "subscriber.py", line 24, in connect_mqtt
    client.tls_set(ca_certs="/home/alfa/mosquitto/certs/ca.crt",
  File "/home/alfa/.local/lib/python3.8/site-packages/paho/mqtt/client.py", line
796, in tls_set
    context.load_cert_chain(certfile, keyfile, keyfile_password)
ssl.SSLError: [SSL] PEM lib (_ssl.c:4046)
alfa@alfa-VirtualBox:~/mosquitto$ python3 subscriber.py
Connected to MQTT Broker!
```

Figura 22: Ejemplo de conexión, suscripción y publicación de los datos

### 6.2.1. Conexión por bróker MQTT

Para simular el envío de datos a través de internet del auditor al centro de control utilizamos la librería Paho MQTT para publicar los mensajes a un bróker en el tópic “python/mqtt/cripto/test”, en el cual después nos vamos a suscribir para poder recibir los datos. Además para la implementación de la comunicación

bidireccional, vamos a crear otro tópico, el cual únicamente será usado para este fin, así nos aseguramos de tener tópicos diferentes y una buena organización.

Lo primero que debemos de hacer es la autenticación del aparato físico que es el auditor con nuestra CA que es nuestro centro de control, con el código que se encuentra en nuestro repositorio de GitHub [18] se crean y firman los certificados y clave que podemos ver en la figura 23, en donde para poder publicar en el canal se necesita estar autenticado con estos certificados previamente solicitados por el auditor y firmados por la autoridad certificadora.

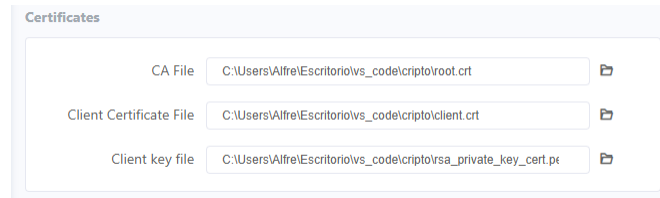


Figura 23: Certificados en dashboard del bróker

### 6.3. Credenciales y archivos iniciales

Además de los certificados, también debemos crear un usuario y una contraseña para cada auditor en el bróker MQTT con las siguientes credenciales que nosotros generamos y establecer conexión con el bróker:

- Usuario: auditor\_1
- Contraseña: poner\_una\_contraseña\_muy\_segura1
- Usuario: auditor\_2
- Contraseña: poner\_una\_contraseña\_muy\_segura2

Asimismo, debemos crear las credenciales correspondientes en el bróker MQTT para la autenticación del centro de control y poder establecer conexión con el bróker:

- Usuario: centro\_control
- Contraseña: poner\_una\_contraseña\_muy\_segura3

En primer lugar, en el auditor se generarán las claves públicas para que el centro de control pueda verificar las firmas. Utilizaremos el siguiente código y enviaremos los resultados:

- Ecdsa\_public\_keys.py
- ECDH\_public\_key.py
- xPublickey\_ecdsa.key
- yPublickey\_ecdsa.key
- xPublickey\_ECDH.key
- xPublickey\_ECDH.key

## 6.4. Encriptado, intercambio seguro de claves y firmado de las trazas

Para enviar y recibir datos, necesitamos llevar a cabo la creación e intercambio de claves públicas, que es una de las primeras etapas de nuestro reto. Para esto, debemos compartir nuestras claves públicas y privadas utilizando el archivo `CC_ECDH_publicKey.py` en el centro de control, y `AU_ECDH_publicKey.py` en el auditor. Ambos elementos intercambiarán sus claves publicándolas en el tópico `/publicKeys`. Después de intercambiar las claves públicas, utilizaremos `CC_ECDH_sharedSecret.py` y `AU_ECDH_sharedSecret.py` para determinar el secreto que ambos elementos compartirán. Este proceso debe realizarse en cada sesión, que durará 8 horas.

A continuación, abordaremos el proceso de intercambio de claves:

Cliente 1 (Emisor):

1. Generar una clave privada (`privKeyA`) y calcular la correspondiente clave pública (`xPublicKeyA`, `yPublicKeyA`) utilizando `EccMultiply`.
2. Mantener la clave privada segura y proporcionar la clave pública al receptor.

Cliente 2 (Receptor):

1. Generar una clave privada (`privKeyB`) y calcular la correspondiente clave pública (`xPublicKeyB`, `yPublicKeyB`) utilizando `EccMultiply`.
2. Mantener la clave privada segura y proporcionar la clave pública al emisor.

Cliente 1 (Emisor):

1. Multiplicar la clave pública del Cliente 2 (`xPublicKeyB`, `yPublicKeyB`) con la clave privada del Cliente 1 (`privKeyA`) utilizando `EccMultiply` para calcular el secreto compartido.
2. Utilizar el secreto compartido como clave para algoritmos de encriptación simétrica (por ejemplo, AES) para encriptar la información deseada.

Cliente 2 (Receptor):

1. Multiplicar la clave pública del Cliente 1 (`xPublicKeyA`, `yPublicKeyA`) con la clave privada del Cliente 2 (`privKeyB`) utilizando `EccMultiply` para calcular el secreto compartido.
2. Utilizar el secreto compartido como clave para algoritmos de encriptación simétrica (por ejemplo, AES) para desencriptar la información encriptada recibida.

En el archivo `ECDH_vectores_de_prueba.ipynb` en el repositorio se muestra un ejemplo de cómo utilizar estos códigos con vectores de prueba y luego encriptarlos usando la clave compartida con AES-256.

```
-----BEGIN ECC PRIVATE KEY-----
MIGHAgEAMBMGBYqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQg6zDErCjgoFRq8MX
Pe7pd5NrV44MOUyGhNXLxyZFBhqhRANCAASKwWxeiztTeUpNKzCMsfaZW31IfmEi
GbLRqafmCjTfd2L1ywBtRN8QbcNGC6UzplcjmlQaHPJou6w62QmoLPR
-----END ECC PRIVATE KEY-----
```

Figura 24: Ejemplo llave privada de ECC

```

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAI6tZOSmtKxua7ZRS0ImpUg24JhShGSIP/khIbuF0TudzfSKg
phy2SmeNKx1BaKqFQCD5Xt0XCufQcASu3ZEScfsGjP0/2lg78K/z8SZrYZUEVJ81
yGD6uwtPT0T0pvipL8LDgQhPIX/OY+FEUPC/lDBK08nU54L74ryMS/18S3UJVJdaE
wOL4WjNWYNJGvGS41r1dL3yrnwBpXBIVT7GLfZ91kmSQyb0miHGkZVmlEfymT0S2
jgILNkkRPetxRsr8ZoI2m+VOIJ+ghf7obqEB7kdmPaUIyvbBRbs2DMWfAoz3M+s+
J89udRS3eGiA4J/j5I+b/gw5TVt9fUjKsFvgfQIDAQABoIABAANHx1a3aRqUTVo
1merjhB+U0R4csI+ZkDz15cy11x1QKdeoSdX/rkbtCxiTPAM1HS+NIPk93j27GBC
rQnQo7fYx41rKpWG5g2ouFp0bhdRMVv65d73/seGJZ3jjC59jUIZU65Rq1kdKpx
X1+db/q3H1N11tT29tSKLoQ0QsK2glRZwdQwz+Fy5TV7YeYB1lwQkDmsmdY9cQ
HhBZJXUM/OvU7qF2uUCei/mRQANNumJnSUYEkXlmmxr+fJF7Z428Q2Qy9mgTnfN0
CULiaabt6x+C+ikn81o1DUEzXgx0dtrBwNICPd/Np6YJBu/NM+wFHg3DSvBM66P6
/Vz/23sCgYEAucjCDI7F3H/fkCm6VJG1UEG+HT9FbhOQLZrSLfipztVNTtjCmp1
76wFkj3sUCFAnWz5bMsGxvNFGLEEDLRQf7nsaWoUi8AJt5xQK8YkVRupP4KEx8
bQltBfuuBKjs0Hc6/VBY7DEigKqLM7/s0yxg3R2jlmZnh/DIX68gG0McGYEAwHTS
b8t+ASRFQM/QkpuZnRQCc1b20gtN1f+kPsZGIiveZFjefWh+bTk3Ld0f0GCN6U2f
lelz0GCU3YCIWtrkwaasohu5IyIY1P+tnvJf83B0icX5FkeY67c1Txblqecx1l
9StGhcbebgR05mKMSn10wbPB5csDZx42n1+b+T8CgYA0/Dezoyh1CZc2I2B77A58
9DMHXLsr2LJ+/y/AYrHsiWwh+GwYdWJrV3FDUxvJgqJOA5JD6udk6E61jGf0zONW
e79ceiDNoNPVjIH/GRenI0P65Waz/65g3pZUZYIs3AymK8sMIUqj5zae0k2Gh+t
3nNFvMVCCLSwRZCwEV8jxwKBgF8VmJ6JQEDaJakGw3qbi8KSjz7kfhRhXzy7p+8n
0KCzYdXchliBVu0ptnxfYod2HMBmhDv/Ts3bHUfOr5LF0P1oFqSk48xa3T/m7mW
4Q4E0a1ViTVVXm+wdkSX5IX+7YfaRYhikB4qCdZ7MGVL2Ro2YtXKtQKOUakRwVRv
0kTLAoGBALBc0bkwwJx1lglc13Ip44f1nv+L9gms0D181PoEkbS/3yUgg8Xju+oi
KwDtrPjLr1h2QAGI4dZ+YC5/STTVTF+pW+hNwoeMKnNqYLUspHzgh2FjakG4Qob2
mqw2K690sPSPH4sIdwL/p4eyXdeIXj2qksh3toDsgseHxpz0UAZ7
-----END RSA PRIVATE KEY-----

```

Figura 25: Ejemplo llave privada de RSA

En la figura 25 podemos ver un ejemplo de la llave privada generada con RSA y en la figura 24 una clave con ECDH-256, en donde podemos ver la gran diferencia entre el tamaño de ambas claves y la razón por la cual usamos ECDH para compartir la clave secreta y no RSA, considerando que usaremos un microcontrolador como auditor.

El orden de ejecución de los códigos es el siguiente:

1. CC\_ECDH\_publicKey.py: Generación de claves ECC en el centro de control.
2. AU\_ECDH\_publicKey.py: Generación de claves ECC en el auditor.
3. CC\_ECDH\_sharedSecret.py: Obtención del secreto compartido con la clave publica en el centro de control.
4. AU\_ECDH\_sharedSecret.py: Obtención del secreto compartido con la clave publica en el auditor.

Una vez que tengamos la clave compartida, podremos comenzar a utilizar la encriptación de los datos de las trazas que se enviarán. Es importante destacar que se debe agregar un vector de inicialización (IV) para cada traza.

```

{'encrypted_data': ".*%0é\x02t1n\x1e)...etc,
'iv': '\x8e\x1f\x88!\x82ðGa\x9e'\x1eV....etc'}

```

Después de encriptar los datos con el algoritmo de encriptación simétrica AES, implementaremos la firma con ECDSA y la agregaremos a cada traza antes de enviarla al centro de control, a continuación un ejemplo de la traza en este punto:

```

{'encrypted_data': ".*%0é\x02t1n\x1e)...etc,

```

```
'iv': '\x8e\x1f\x88!\x82òGa\x9e'\x1eV\....etc',  
'r': '0x5e43d3f6326149fd7ea4aa946b66e7....etc',  
's': '0x66ffad681da4a744b12d56a671a156....etc'}
```

Una vez que las trazas estén encriptadas y firmadas con sus respectivos IV, r y s, podremos trabajar con el centro de control. A continuación, abordaremos el proceso inverso:

1. Conectamos al auditor al tópico utilizando el código previamente configurado con las credenciales.
2. Al recibir los mensajes, lo primero que haremos será verificar la firma con curvas elípticas.
3. En la verificación de las firmas, extraemos los datos relevantes y verificamos la firma. Si la firma es válida, almacenamos los datos en la lista 'valid\_traces'.

Después de validar las trazas, procederemos a descryptar los datos utilizando la clave compartida previamente. Ahora tendremos los datos descryptados y listos para su procesamiento en el centro de control. Luego de esto, continuaremos con el envío de los datos a la base de datos. A continuación un ejemplo de los datos descryptados y listos para ser analizados o almacenados:

```
{'trace_id': '011',  
'timestamp': '2013-11-02T00:00:00Z',  
'C(0)/P(1)': '0',  
'value': 58.0}
```

## 6.5. Creación de base de datos e inserción de datos

Para la base de datos, utilizaremos nuestro propio servidor con MySQL. A continuación, describiremos los pasos para configurar la base de datos y realizar las operaciones necesarias:

1. Levantamos nuestro servidor con WampServer.
2. Nos autenticamos en myphpAdmin, usuario: root, password:
3. Creamos una tabla que contenga los campos de las trazas.
4. Creamos una base de datos llamada 'cripto\_test'.
5. Luego, creamos una tabla donde insertaremos las lecturas obtenidas. Al momento de crear la tabla, se debe cambiar la configuración del 'Storage Engine' a CSV.
6. Usamos el código SQL\_data\_insert.py e insertamos las trazas que van llegando a nuestra base de datos como se ve en la siguiente figura.

## 7. Conclusiones y trabajo futuro

En conclusión, este proyecto ha abordado el desafío de proteger las comunicaciones y el almacenamiento de datos en entornos de Internet de las Cosas utilizados para monitorear la producción y el consumo de energía. Hemos propuesto un esquema de intercambio de datos que garantiza la seguridad de la información, preservando la confidencialidad, integridad, autenticidad y privacidad de los datos. Para lograr esto, hemos implementado una arquitectura de red sólida y un protocolo criptográfico seguro que asegura que los datos sensibles, como los patrones de consumo y producción, no sean comprometidos en ningún momento, desde su generación hasta su almacenamiento.

Showing rows 0 - 9 (10 total. Query took 0.0049 seconds)

```
SELECT * FROM `base_datos_cc`
```

☐ Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

trace_id	timestamp	c. or p	value
11	2013-11-02T00:00:00	0	58
12	2013-11-02T00:15:00	0	75
13	2013-11-02T00:30:00	0	65
14	2013-11-02T00:45:00	0	0
15	2013-11-02T01:00:00	0	67
16	2013-11-02T01:15:00	0	69
17	2013-11-02T01:30:00	0	0
18	2013-11-02T01:45:00	0	73
19	2013-11-02T02:00:00	0	68
110	2013-11-02T02:15:00	0	0

☐ Show all | Number of rows: 25 | Filter rows: Search this table

Query results operations

Figura 26: Base de datos SQL en myphpAdmin con datos insertados

Este proyecto ha demostrado la importancia de la implementación de esquemas criptográficos en entornos de IoT para proteger la información crítica. Al salvaguardar los datos, aseguramos la confianza y la confidencialidad de los usuarios y evitamos posibles amenazas y vulnerabilidades en el sistema.

En última instancia, nuestro enfoque en la seguridad de los datos en el contexto del monitoreo de la energía ha contribuido a mejorar la confiabilidad y eficiencia de los sistemas fotovoltaicos en diferentes entornos, desde residencias hasta instalaciones industriales. La implementación exitosa de nuestro esquema criptográfico ha sentado las bases para futuros desarrollos y avances en la protección de la información en la era de la Internet de las Cosas.

Como un trabajo a futuro que recomendamos al socio formador están las siguientes recomendaciones basadas en nuestra propuesta de solución:

1. Implementación de un sistema de backup utilizando un servidor AAA (Authentication, Authorization, and Accounting) para garantizar la seguridad y disponibilidad de los datos, para esto se puede utilizar AWS Radius.
2. Aplicación de una instancia de almacenamiento en la nube para la base de datos como AWS S3.
3. Utilización de un bróker MQTT en la nube como EQMX para asegurar la escalabilidad y viabilidad del proyecto, permitiendo una comunicación eficiente y confiable entre los dispositivos y el centro de control.
4. Implementación de una red virtualizada que conecta de manera eficiente y segura smart sensors con un centro de control, permitiendo la comunicación y gestión centralizada de los dispositivos para un monitoreo y control efectivo.

Se debe considerar que algunas de estas conllevan costos extras que deben ser considerados, sin embargo, tienen ventajas importantes que también deben explorarse, esto sin necesidad de modificar la arquitectura base de la solución propuesta.

## 8. Anexos


### 8.1. Vectores de Prueba

Para comenzar a trabajar con la base de datos vamos a utilizar vectores de prueba, los cuales van a tener la estructura de un diccionario para facilitar el manejo y envío de estos a través del protocolo MQTT, este es el ejemplo de una traza descriptada:

```
{"trace id": "1", "timestamp": "2013-11-02T00:00:00Z", "C(0)/P(1)": "0", "value": 58.0},
```

luego seguimos el proceso para compartir los datos entre el auditor y el centro de control:

1. Primero, se configuran variables importantes, como el nombre de dominio del bróker MQTT, el número de puerto y los temas de publicación y suscripción.
2. A continuación, se define una función llamada *connect\_mqtt()* que crea una instancia del cliente MQTT y se conecta al bróker MQTT. Esta función también define una función de devolución de llamada que se ejecuta cuando el cliente MQTT se conecta con éxito o falla en la conexión al bróker.
3. Después de definir la función *connect\_mqtt()*, se crea una instancia del cliente MQTT utilizando un ID de cliente aleatorio y se asigna la función de devolución de llamada.
4. A continuación, se llama a la función *connect\_mqtt()* para establecer la conexión con el bróker MQTT.
5. Luego, se define una función llamada *publish()* que toma como argumentos el cliente MQTT y los datos que se desean publicar. Esta función se encarga de publicar los mensajes MQTT en intervalos regulares.
6. Dentro de la función *publish()*, se inicia un bucle infinito para simular la publicación continua de mensajes. En cada iteración del bucle, se pausa la ejecución del programa durante 15 minutos antes de publicar el siguiente mensaje.
7. Los datos que se desean publicar se convierten en formato JSON utilizando el módulo *json* antes de ser publicados mediante el cliente MQTT.
8. Después de publicar un mensaje, se verifica el resultado de la publicación y se muestra un mensaje de éxito o error en la consola, dependiendo del estado de la publicación.
9. Por último, se crean ejemplos de trazas en forma de diccionarios en Python, cada uno representando una traza con información como un ID, una marca de tiempo y un valor. Se crea una instancia del cliente MQTT utilizando la función *connect\_mqtt()* y se inicia el bucle de eventos del cliente MQTT para mantener la conexión activa.



```
Connected to MQTT Broker!  
Published message 1 of 10  
Published message 2 of 10  
Published message 3 of 10  
Published message 4 of 10  
Published message 5 of 10  
Published message 6 of 10  
Published message 7 of 10  
Published message 8 of 10  
Published message 9 of 10
```

Figura 27: Conexión del auditor al servidor MQTT por el publisher

En la imagen 27, podemos ver como se establece la conexión al servidor MQTT de mosquitto y también podemos ver como obtenemos el mensaje de verificación cada vez que la publicación de una traza es exitosa,

hay que mencionar que esta vez usamos trazas que nosotros elegimos para enviar periódicamente cada 5 segundos para cuestiones practicas, luego de que el auditor comienza a publicar las trazas tenemos que el centro de control las recibe, al estar suscrito al mismo tópico en el que son publicadas:

```

Connected to MQTT Broker!
Received '{"encrypted_data": "u00beu00b3u00e9u0002tInu001e)u00b3u00d4u00e0u0015u00a2cou00fcu00bfcu00ba3u00e9fu00bcu00d09ZAu00dbu00f5u00d5u0000u0012u0006u00c0?u00eefu0015j'u00a4"}'
Received '{"encrypted_data": "Inu00b8u009d'u00b0b6$u00b9u0008u001u00fa"u0000u00eeu00bcu00a3u0016u0088u00ba2x'u0011u0087f'u0009u00ad'u00deu00a2u00a8u008fh'u00d6u00a7u0091u00ab'u0002"}'
Received '{"encrypted_data": "AU00d5u00d4u00d7u00u00c3u00f9u00c5'u0007u00d6u00bbbu00ba0u0083u00b6u00ba0u0001u00b4u00a0u008Hu00eb'u0012u0009u007ZpL'u00e6u00f0P'u00f9u00a36'u00bf"}'
Received '{"encrypted_data": "8E\u00f0F\u0013u00a9l'u00d2u0018"BV\u0010u0000u0000u0000u0000u0000u0094u007fu00bcu00d2u00d8cu00fdZu00d0u00fcu00bcu00f1u0000u00d6u001btf'u00e4c"u00c9c\u0009u00d8"}'
Received '{"encrypted_data": "u00ddu00d3u00bfu001fp'u0005u0008g\u00a3u00f9t\u00c0u0094u0007du00ff\u0007u0008;u00c2u0016u00d0u00ec'u00a4q\u00c0u0094u0081u0085-u00eews\u00f0d\u00dbd\u00bf"}'
Received '{"encrypted_data": "u0015u00ffu001a\u00b9u0002u00dcg\u0010u0012u00a7u001au00aa"u00bb0u0093u0094u00dfd\u0098p\u00d1u00a7u00d4-s\u0017u0019u001eu00f6u00d16u00f0u0080"u0004?u0009"}'
Received '{"encrypted_data": "u00812u00dbu00b40A"u0087-u00ddu0098u00eak\u00d5H\u0018u00a28\u0085q\u008ec\u00dau0015u00fd\u00a3u00e60u00d8u00ea\u0007u00deL\u00c8u00d7u00eb"u00aeu00e"}'
Received '{"encrypted_data": "u009c\u00d5u00cc\u00tVl\u00dcu009ezu"u008e>-u00fd\u00b2u0094u003Hu0094u0099u0096\u00b5u00edf\u0000u00d70u001du00160u00fcu00ae-u0095u00f2"u00bb-b\u00b"}'
Received '{"encrypted_data": "u00ec\u0094u0090u00f0u00f6u008f\u0008\u00bbWQ5u00d4u0000qu0001DK\u0002u00da\u00c5u00d0u0006"i\u00b7u00c7u00d5u0093u00e8\u0098u00ea\u00abqc\u00e2u00fa\u00ad\u00f3"}'
Received '{"encrypted_data": "-u00a09h\u0004u00fbo\u00dbu00a10;C\u000fu00ef\u0009e\u0099u00cCa\u001eu0098u00f3u0007u001au00b2u008cc\u0091u00e8u00b3u0000u00f5u0009b\u00dbd\u00a5u001bu0092u00"}'

```

Figura 28: Vectores de prueba recibidos por el subscriber (CC)

En esta figura 28, podemos ver como el centro de control que está suscrito al mismo t pico en el que est n siendo publicadas las trazas las recibe sin mayor complicaci n, esto funciona a pesar de estar conectados en 2 redes diferentes a trav s del servidor MQTT de mosquitto, podemos ver que nuestros 10 vectores de prueba llegan intactos y completamente funcionales, cabe recalcar que estos datos se muestran encriptados y con la firma con ECDSA, para referencia se puede consultar el repositorio de GitHub para ver el c digo completo. [18]

Luego de que recibimos las trazas firmadas vamos a verificar que la firma sea autentica y los datos no hayan sido modificados en el transito, hay que recalcar que en esta parte solamente nos quedaremos con los datos que hayan pasado el proceso de verificación de la firma, es decir, en donde se obtenga Valid Signature, que en este caso en la figura 29 fueron todos porque no los modificamos.

[illegible]

Figura 29: Verificación de la firma de los vectores de prueba



Después de verificar las firmas procedemos a descriptar los datos con el secreto compartido que obtuvimos al usar ECDH que podemos ver en la figura 30

```
key = bytes.fromhex('05bb88dde58728d36ef20a116decd35cdda24a4c2798050074d76c23fc47a73') #llave compartida pr

decrypted_data_valid_traces = []
# Decrypt and print each trace
for encrypted_trace in valid_traces:
    ciphertext = encrypted_trace['encrypted_data'].encode('latin-1')
    iv = encrypted_trace['iv'].encode('latin-1')
    decrypted_data = decrypt_data(key, iv, ciphertext)
    decrypted_data_valid_traces.append(decrypted_data.decode('latin-1'))
    print("Decrypted Data:", decrypted_data.decode())

Decrypted Data: "trace_id": "011", "timestamp": "2013-11-02T00:00:00Z", "C(0)/P(1)": "0", "value": 58.0
Decrypted Data: "trace_id": "012", "timestamp": "2013-11-02T00:15:00Z", "C(0)/P(1)": "0", "value": 75.0
Decrypted Data: "trace_id": "013", "timestamp": "2013-11-02T00:30:00Z", "C(0)/P(1)": "0", "value": 65.0
Decrypted Data: "trace_id": "014", "timestamp": "2013-11-02T00:45:00Z", "C(0)/P(1)": "0", "value": 0.08
Decrypted Data: "trace_id": "015", "timestamp": "2013-11-02T01:00:00Z", "C(0)/P(1)": "0", "value": 67.0
Decrypted Data: "trace_id": "016", "timestamp": "2013-11-02T01:15:00Z", "C(0)/P(1)": "0", "value": 69.0
Decrypted Data: "trace_id": "017", "timestamp": "2013-11-02T01:30:00Z", "C(0)/P(1)": "0", "value": 0.07
Decrypted Data: "trace_id": "018", "timestamp": "2013-11-02T01:45:00Z", "C(0)/P(1)": "0", "value": 73.0
Decrypted Data: "trace_id": "019", "timestamp": "2013-11-02T02:00:00Z", "C(0)/P(1)": "0", "value": 68.0
Decrypted Data: "trace_id": "0110", "timestamp": "2013-11-02T02:15:00Z", "C(0)/P(1)": "0", "value": 0.06
```

Figura 30: Descriptación de los datos encriptados de los vectores de prueba

Y finalmente publicamos los datos en nuestra base de datos de SQL como podemos ver en 31 y podemos ver el resultado en la base de datos creada anteriormente en la figura 32.

```
# Commit the changes to the database
mydb.commit()

print(mycursor.rowcount, "record(s) inserted.")

1 record(s) inserted.
```

Figura 31: Inserción de los vectores de prueba en la base de datos

Showing rows 0 - 9 (10 total. Query took 0.0049 seconds)

SELECT \* FROM "base\_datos\_cc"

☐ Profiling ☐ Edit inline ☐ Edit ☐ Explain SQL ☐ Create PHP code ☐ Refresh

☐ Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

trace_id	timestamp	c	nr	p	value
11	2013-11-02T00:00:00Z	0			58
12	2013-11-02T00:15:00Z	0			75
13	2013-11-02T00:30:00Z	0			65
14	2013-11-02T00:45:00Z	0			0
15	2013-11-02T01:00:00Z	0			67
16	2013-11-02T01:15:00Z	0			69
17	2013-11-02T01:30:00Z	0			0
18	2013-11-02T01:45:00Z	0			73
19	2013-11-02T02:00:00Z	0			68
110	2013-11-02T02:15:00Z	0			0

☐ Show all | Number of rows: 25 | Filter rows: Search this table

Figura 32: Base de datos SQL en myphpAdmin con datos insertados

## Referencias

- [1] “Types of Network Architecture,” 1 2018.
- [2] “P2P: ARQUITECTURA DE RED PEER-TO-PEER,” 2 2021.
- [3] MLSDev, “Blockchain architecture basics: Components, structure, benefits creation,” 2019.
- [4] “¿Qué es la arquitectura de red y cuáles son sus funciones?,” 1 2021.
- [5] “Arduino vs Raspberry: Whats the difference?,” 10 2022.
- [6] C. S. Inc, “WLSE Express AAA Server Certificate Configuration Guide,” 5 2005.
- [7] J. Yang, J. Fan, and X. Zhu, “Perception layer lightweight certificateless authentication scheme for iot-based emergency logistics,” *IEEE Access, Access, IEEE*, vol. 11, pp. 14350 – 14364, 2023.
- [8] K. Shim, “Design principles of secure certificateless signature and aggregate signature schemes for iot environments.,” *IEEE Access, Access, IEEE*, vol. 10, pp. 124848 – 124857, 2022.
- [9] A. D. J. C. BARRAZA, “Modelo de implementacion de ciberseguridad para sistemas iot en el marco de redes 5g,” 2019.
- [10] N. Naik, “Choice of effective messaging protocols for iot systems: MQTT, CoAP, AMQP and HTTP,” 2017.
- [11] B. Wukkadada, K. Wankhede, R. Nambiar, and A. Nair, “Comparison with HTTP and MQTT in internet of things (IoT),” *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, 2018.
- [12] “CryptoSys PKI Pro,” 1 2023.
- [13] M. Al-Zubaidie, Z. Zhang, and J. Zhang, “Efficient and secure ecDSA algorithm and its applications: A survey,” *arXiv preprint arXiv:1902.10313*, 2019.
- [14] X. Fan, “Security Challenges in Smart-Grid Metering and Control Systems,” 2013.
- [15] “HUAWEI WiFi AX2 Smart Router, Wi-Fi 6+, 1500 Mbps, 2.4ghz 5ghz, Blanco : Amazon.com.mx: Electrónicos.”
- [16] “HUAWEI WiFi AX2 Smart Router, Wi-Fi 6+, 1500 Mbps, 2.4ghz 5ghz, Blanco : Amazon.com.mx: Electrónicos.”
- [17] Amazon, “Precios de amazon s3,” s.f.
- [18] J. A. Garcia, “[https://github.com/alfredogrcaa/cripto\\_reto\\_licore](https://github.com/alfredogrcaa/cripto_reto_licore).”