

Trabajo Fin de Grado  
Grado en Ingeniería de las Tecnologías de  
Telecomunicación

Encriptación y cifrado de datos en plataformas IOT  
basadas en FIWARE

Autor: Guillermo Martínez Herrera

Tutor: Jorge Calvillo Arbizu

Dpto. Ingeniería Telemática  
Escuela Técnica Superior de Ingeniería  
Universidad de Sevilla

Sevilla, 2021





Trabajo Fin de Grado  
Grado en Ingeniería de las Tecnologías de Telecomunicación

# **Encriptación y cifrado de datos en plataformas IOT basadas en FIWARE**

Autor:

Guillermo Martínez Herrera

Tutor:

Jorge Calvillo Arbizu

Profesor Ayudante Doctor

Dpto. de Ingeniería Telemática  
Escuela Técnica Superior de Ingeniería  
Universidad de Sevilla

Sevilla, 2021



Trabajo Fin de Grado: Encriptación y cifrado de datos en plataformas IOT basadas en FIWARE

Autor: Guillermo Martínez Herrera

Tutor: Jorge Calvillo Arbizu

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal



*A mi familia*  
*A mis amigos*





# Agradecimientos

---

Primeramente, a mi madre por seguir apostando por mí y mi futuro laboral, a pesar de mis tropiezos y mi pasividad para afrontar la carrera.

A Eu, Luis, Sergio, Domin y Javi, que empezamos juntos la carrera, y aunque cada uno hemos diversificado nuestros objetivos, han sido un apoyo para seguir y siempre me han prestado su ayuda cuando la he necesitado, además de grandes amigos.

Debo agradecer a mi tutor el compromiso adquirido conmigo para desarrollar este trabajo. Él supo encontrarme una oportunidad sobre un tema que me gustaba como es la seguridad. Haber podido aprender sobre FIWARE también me ha resultado gratificante y ahora valoro mucho más las salidas laborales en IOT.

*Guillermo Martínez Herrera*

*Sevilla, 2021*



En la actualidad, la necesidad del hombre por controlar todo y el desarrollo de las tecnologías, nos permite conectar múltiples cosas a nuestro alrededor por internet. A esto le llamamos *Internet Of Things*.

A través de este proyecto se va a trabajar basándonos en una plataforma IOT que interconecta datos sanitarios adquiridos de sensores en camisetas de pacientes con sus respectivos doctores. Esta plataforma estará basada en FIWARE, que es un proyecto impulsado por la UE para fomentar el *Internet del Futuro*.

Esta idea tiene una tarea pendiente, y se trata de la securización de las comunicaciones en ella, algunos agentes ajenos podrían intentar interceptar los datos de los pacientes. Por eso, en este proyecto vamos a estudiar una manera para mejorar este punto débil, y se trata de la implementación de la encriptación y cifrado de datos de estas comunicaciones. Sin embargo, esta metodología tiene la problemática de que normalmente los dispositivos suelen ser de bajo consumo energético y requerirán las menores tareas posibles. Por tanto, analizaremos si podrán llevarlo a cabo.

El proyecto estará basado en dos escenarios diferentes para la encriptación, uno en el que emplearé el mecanismo AES para cifrar un valor y otro en el que será el protocolo HTTPS el que encripte la transmisión completa de información, para posteriormente poner en práctica el segundo. Esta prueba experimental la vamos a llevar a cabo en un entorno virtualizado localmente.

Por tanto, se llevará a cabo la puesta a punto de la plataforma y se explicará su funcionamiento para, a partir de ahí, llevar a cabo las modificaciones necesarias para implementar la encriptación con éxito. Esto lo comprobaremos con la herramienta Wireshark de captura de paquetes, con la que posteriormente también podremos llevar a cabo un estudio del tiempo extra que nos supondrá llevar a cabo la encriptación.

La realización de este proyecto ha supuesto de una carga de trabajo importante a la hora de llevar a cabo la implementación de la plataforma basada en FIWARE, ya que sus documentos no han sido suficientemente intuitivos o con mis medios me he encontrado limitado. Por eso creo que sería bueno que este trabajo se pueda avanzar más adelante, con diferentes medios y más tiempo. Existen muchas mejoras posibles: poder obtener una IP fija para tener un certificado digital firmado por una AC de confianza y desde la que se pueda acceder no solo desde nuestra red local, poder montar la plataforma bajo sensores y dispositivos físicos y llevar a cabo las medidas de encriptación correctamente aplicadas, o la inclusión de más componentes de la plataforma FIWARE que ayudarían a tener una inmersión completa en la plataforma securizada con Generic Enablers como Wilma PEP Proxy, Keyrock o Authforce.

# Abstract

---

Nowadays, man's need to control everything and the development of technologies, allows us to connect multiple things around us through the internet. We call this *Internet Of Things*.

Through this project, we will work on the basis of an IOT platform that interconnects health data acquired from sensors in patients' shirts with their respective doctors. This platform will be based on FIWARE, which is a project driven by the EU to promote the *Internet of the Future*.

This idea has a pending task, and that is the securization of communications in it, some external agents could try to intercept patient data. Therefore, in this project we are going to study a way to improve this disadvantage, and it is the implementation of encryption and data encryption of these communications. However, this methodology has the problem that the devices are usually low power consumption and will require as few tasks as possible. Thus, we will analyze whether it will be worthwhile to implement it.

The project will be based in two different scenarios for encryption, one in which I will use the mechanism AES to encrypt a value and another in which the protocol HTTPS will encrypt the whole transmission of information, and then implement the second one. This experimental test will be carried out in a locally virtualized environment.

Therefore, the platform will be set up and its operation will be explained in order to carry out the necessary modifications to implement the encryption successfully. This will be checked with the Wireshark packet capture tool, with which we will also be able to carry out a study of the extra time it will take to carry out the encryption.

The realization of this project has meant a significant workload when carrying out the implementation of the platform based on FIWARE, as their documents have not been sufficiently intuitive or with my means I have been limited. That is why I think it would be good if this work could be advanced further, with different means and more time. There are many possible improvements: to be able to obtain a fixed IP to have a digital certificate signed by a trusted CA and from which it can be accessed not only from our local network, to be able to mount the platform under sensors and physical devices and carry out the encryption measures correctly applied, or the inclusion of more components of the FIWARE platform that would help to have a complete immersion in the platform secured with Generic Enablers such as Wilma PEP Proxy, Keyrock or Authforce.

<b>Agradecimientos</b>	<b>9</b>
<b>Resumen</b>	<b>11</b>
<b>Abstract</b>	<b>12</b>
<b>Índice</b>	<b>13</b>
<b>Índice de Tablas</b>	<b>15</b>
<b>Índice de Ilustraciones</b>	<b>16</b>
<b>Notación</b>	<b>18</b>
<b>1 Introducción</b>	<b>20</b>
1.1 <i>Motivación</i>	20
1.2 <i>Objetivos</i>	20
1.3 <i>Solución planteada</i>	21
1.4 <i>Plan de trabajo</i>	23
<b>2 Estado del arte</b>	<b>25</b>
2.1 <i>Internet Of Things</i>	25
2.2 <i>FIWARE</i>	26
2.2.1 Orion Context Broker	28
2.2.2 IOT Agent-JSON	28
2.3 <i>Encriptación</i>	29
2.3.1 Criptografía Moderna: Cifrado simétrico y asimétrico	29
2.3.2 AES	30
2.3.3 HTTPS	31
2.4 <i>Herramientas utilizadas</i>	32
2.4.1 Máquina Virtual e imagen Ubuntu 20.04 LTS	32
2.4.2 MongoDB	33
2.4.3 cUrl	33
2.4.4 Wireshark	33
2.4.5 openssl	34
<b>3 RESULTADOS</b>	<b>35</b>
3.1 <i>Ejecución de componentes de la plataforma basada en FIWARE</i>	35
3.1.1 Instalación y ejecución de Orion	35
3.1.2 Instalación y ejecución de IOT Agent	37
3.2 <i>Análisis teórico de los mecanismos de encriptación</i>	39
3.2.1 Análisis de la aplicación de AES a la plataforma IOT	39
3.2.2 Análisis de la aplicación de HTTPS en la plataforma IOT	40
3.3 <i>Implementación del mecanismo de encriptación HTTPS en la Plataforma IOT</i>	41
3.3.1 Modificación de Orion	41
3.3.2 Modificación de IOTA	42
3.3.3 Demostración de uso de la encriptación	42
3.3.4 Comparativa de uso de recursos respecto a escenario sin encriptación	50
<b>4 Conclusiones y líneas futuras</b>	<b>53</b>

**5 Anexo A: Instalación Orion Context Broker**

**54**

**Referencias**

**57**

# ÍNDICE DE TABLAS

---

Tabla 1-1 - Tarea 1: Análisis y búsqueda de información.	23
Tabla 1-2 - Tarea 2: Análisis y diseño de la plataforma.	23
Tabla 1-3 - Tarea 3: Implementación de la encriptación en la plataforma.	24
Tabla 1-4 - Tarea 4: Pruebas y validación.	24
Tabla 1-5 - Tarea 5: Documentación.	24
Tabla 3-1 - Pasos para la Encriptación y Decriptación mediante AES.	30
Tabla 3-2 – Tiempos conseguidos para la actualización de contexto sin encriptación	50
Tabla 3-3 – Tiempos conseguidos para la actualización de contexto con encriptación mediante HTTPS	51

# ÍNDICE DE ILUSTRACIONES

---

Ilustración 1 – Esquema de la Publicación de datos	21
Ilustración 2 – Esquema consumición de datos	22
Ilustración 3 – Esquema del escenario de encriptación AES	22
Ilustración 4 – Esquema del escenario de encriptación HTTPS	23
Ilustración 5- Internet Of Things	26
Ilustración 6 - Logotipo oficial de FIWARE	27
Ilustración 7 - Esquema de contextos de información basados en entidades	28
Ilustración 8-Esquema de interconexión del IOT Agent	28
Ilustración 9 - Esquema de funcionamiento de cifrado simétrico	29
Ilustración 10 - Esquema de funcionamiento de cifrado asimétrico	30
Ilustración 11 - Esquema de ejemplo de uso de HTTPS	31
Ilustración 12 -Vmplayer Workstation corriendo máquina virtual Ubuntu	32
Ilustración 13 - Fichero JSON que gestiona MongoDB	33
Ilustración 14 - Logotipo oficial de la librería cUrl	33
Ilustración 15 - Ejemplo de captura de paquetes del programa Wireshark	34
Ilustración 16 - Logotipo oficial de la librería openssl	34
Ilustración 17 - Esquema de la plataforma IOT basada en FIWARE que se va a implementar	35
Ilustración 18 - Entidad 'Ejemplo5' creada	36
Ilustración 19 - Comprobación del arranque y funcionamiento de IOT AGENT	39
Ilustración 20-Esquema del escenario 1	40
Ilustración 21- Esquema de descripción del escenario 2	40
Ilustración 22- Simulación del escenario	43
Ilustración 23 - Fichero de configuración de IOTA para el caso HTTP	44
Ilustración 24-Petición de Consulta de Entidades a Orion	44
Ilustración 25 - Petición de consulta captada por Wireshark	45
Ilustración 26- Simulación de Modificación del contexto por parte del sensor	45
Ilustración 27-Captura del paquete que envía el valor que actualiza el sensor	46
Ilustración 28 - Modificación del fichero de configuración de IOTA para HTTPS	47
Ilustración 29 - Petición de Consulta a OCB mediante HTTPS	47
Ilustración 30 - Captura mediante Wireshark de un paquete de intercambio de información en la Consulta a OCB	48
Ilustración 31 - Simulación de Modificación del contexto por parte del sensor	48
Ilustración 32 - Petición que simula al sensor capturada mediante Wireshark	49
Ilustración 33 - Comunicación entre IOTA y OCB, capturada mediante Wireshark	49



Ilustración 34 - Captura de paquetes mediante Wireshark del proceso de actualización de contexto sin encriptar  
50

Ilustración 35 - Captura de paquetes mediante Wireshark del proceso de actualización de contexto encriptado  
mediante HTTPS  
51

# Notación

---

IOT	Internet Of Things
AES	Advanced Encryption Standard
BBDD	Bases de datos
HTTPS	Hypertext Protocol Secured
OCB	Orion Context Broker
IOTA	IOT Agent
TLS	Transport Layer Security
SSL	Secure Sockets Layer



# 1 INTRODUCCIÓN

---

**E**n este primer apartado se presenta el punto de partida del trabajo, sus objetivos, así como el plan de trabajo. La intención es describir brevemente el alcance y la relevancia del trabajo.

## 1.1 Motivación

En los últimos años estamos siendo testigos de cómo la tecnología está revolucionando todos los sectores de la sociedad, incluido el sanitario.

Uno de los nuevos paradigmas tecnológicos es *Internet Of Things* que posibilita la interconexión entre dispositivos, haciendo uso para ello de la red de Internet, para así poder facilitar el intercambio de datos. IOT pone a disposición de los consumidores una gran cantidad de información publicada por estos componentes autónomos. La llegada del 5G ha permitido que su uso se pueda extender para poder revolucionar el futuro.

El amplio número de agentes que pueden estar implicados en una plataforma IoT y el masivo intercambio de información entre ellos posibilita que terceros maliciosos en la red intenten acceder a ellos por múltiples razones.

Por ello, es necesario garantizar la seguridad de las comunicaciones a través de mecanismos de encriptación y cifrado, entre otros. Sin embargo, el obstáculo de IOT para desplegar este tipo de soluciones es la falta de recursos, ya que se trata de una tecnología que hace uso de sensores y dispositivos de bajo consumo energético para poder obtener la mayor autonomía y vida útil en estos. Es por eso por lo que queremos medir cuánta tarea supondría aplicar la encriptación.

En el ámbito sanitario, IoT puede aplicarse tanto para almacenar datos de pacientes, obtenidos por diversos tipos de sensores con los que controlamos la salud de una persona- por ejemplo, un reloj inteligente recoge datos sobre su ritmo cardiaco-, como para que los médicos que lo requieran puedan obtener al instante esos datos.

La información que se intercambia en estos casos es de salud de los pacientes, la cual tiene un contenido tan sensible que garantizar la seguridad y privacidad de datos y pacientes es aún más acuciante y obligado. Además, los datos de los pacientes están amparados por la ley de protección de datos.

Por todo ello, la motivación de este trabajo es analizar varios mecanismos de encriptación y cifrado en un escenario sanitario donde una plataforma IOT gestiona datos de monitorización de variables fisiológicas de pacientes recopiladas por sensores.

## 1.2 Objetivos

El principal objetivo de este trabajo es implementar un método de cifrado de la información que envía y recibe la plataforma IOT (basada en los componentes FIWARE), para así asegurar la confidencialidad de los datos.

Para ello, primero estudiaremos dos mecanismos de encriptación con el fin de poder elegir qué método es el más adecuado para nuestros escenarios. Analizaremos la seguridad de ambos métodos, para finalmente ver la viabilidad de implementarlos.

Para ponerlos en práctica necesitaremos modificar la plataforma que vamos a usar, basada en FIWARE, con el fin de poder realizar un intercambio de datos totalmente cifrados. La idea es que sea una modificación sencilla y fácil de implementar para así poder ser distribuida.

### 1.3 Solución planteada

Se va a implementar una plataforma IOT basada en FIWARE -localmente, simulando un escenario distribuido-, compuesta de varios sus componentes en el siguiente capítulo, entre los que destacan dos: IOT Agent y Orion Context Broker.

El IOT Agent es la aplicación encargada de recoger datos de los sensores y enviarla al “almacén de información” mediante internet. Ese almacén será el Orion Context Broker (OCB), que guarda contextos de información en una base de datos.

Asimismo, si pensamos en OCB como en un servidor, éste tendrá dos tipos de clientes: el publicador de la información y el consumidor de ella. La publicación de datos está mecanizada en el IOT Agent, la consumición de ellos dependerá sin embargo plenamente del usuario y no será necesaria ninguna aplicación FIWARE, aunque sí una petición correcta.

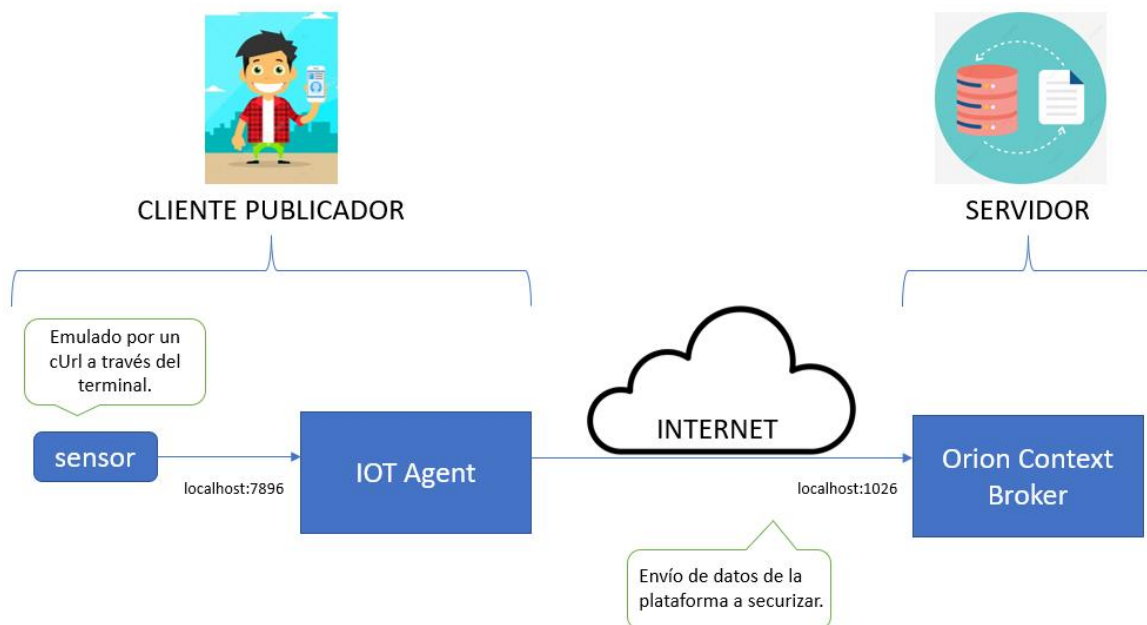


Ilustración 1 – Esquema de la Publicación de datos

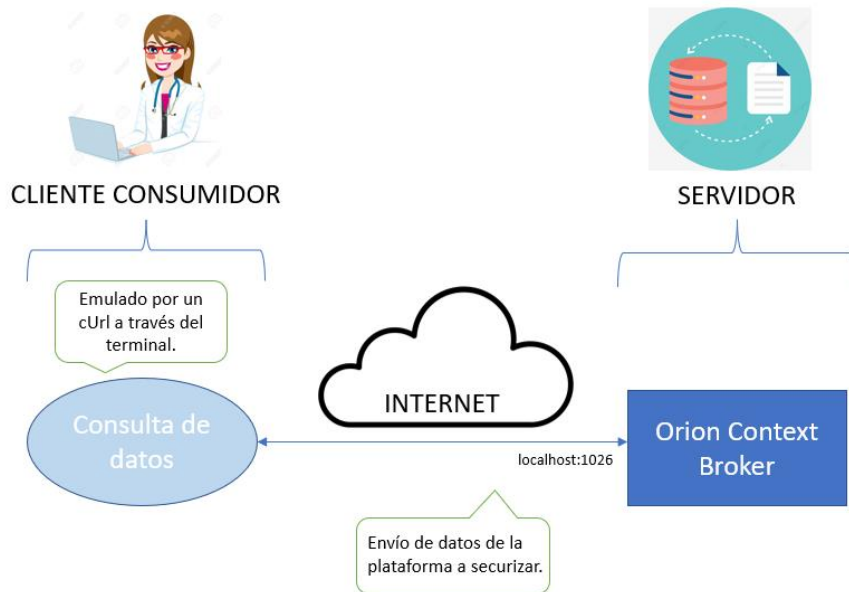


Ilustración 2 – Esquema consumición de datos

A la hora de realizar el envío de datos por Internet hacia el OCB se suelen realizar en texto plano, por ello vamos a analizar dos métodos diferentes de encriptación con el que poder solucionar el problema de la confidencialidad.

El primero será una encriptación en AES del valor que queremos manipular:

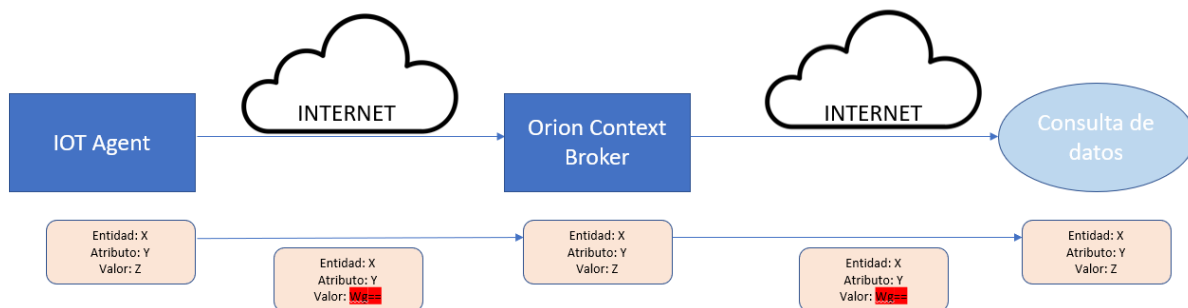


Ilustración 3 – Esquema del escenario de encriptación AES

La segunda solución propuesta será la encriptación del canal mediante HTTPS:

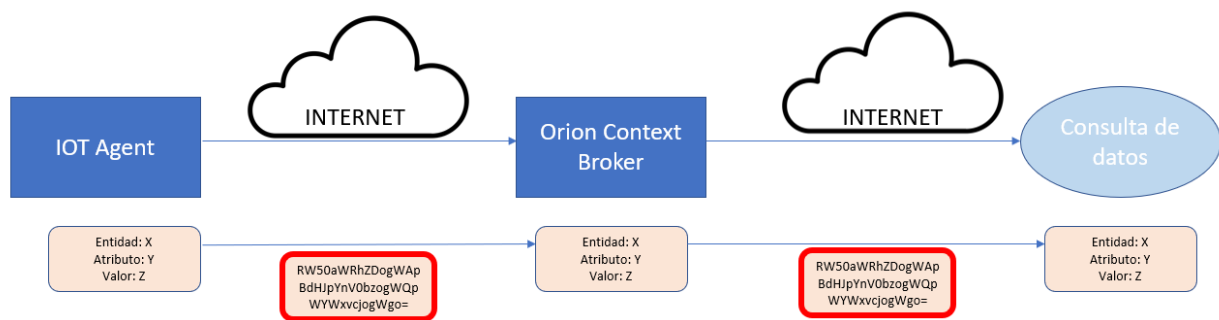


Ilustración 4 – Esquema del escenario de encriptación HTTPS

Para terminar, se implementará el método más adecuado y se realizarán unas pruebas que expliquen su viabilidad y su conveniencia. Todo el escenario será simulado en una máquina virtual Ubuntu.

## 1.4 Plan de trabajo

En este apartado se describe cómo se ha organizado el trabajo y el tiempo que ha requerido llevarlo a cabo.

Subtarea	Tiempo estimado	Tiempo real
Métodos de Encriptación	3h	2h
Internet Of Things	5h	5h
Plataforma FIWARE	5h	7h
<b>Total</b>	13h	14h

Tabla 1-1 - Tarea 1: Análisis y búsqueda de información.

Subtarea	Tiempo estimado	Tiempo real
Instalación de OCB	10h	30h
Instalación de IOTA	5h	10h
<b>Total</b>	15h	40h

Tabla 1-2 - Tarea 2: Análisis y diseño de la plataforma.

<b>Subtarea</b>	<b>Tiempo estimado</b>	<b>Tiempo real</b>
Encriptación en OCB	5h	15h
Encriptación en IOTA	5h	10h
Pruebas	25h	25h
<b>Total</b>	<b>35h</b>	<b>50h</b>

Tabla 1-3 - Tarea 3: Implementación de la encriptación en la plataforma.

<b>Subtarea</b>	<b>Tiempo estimado</b>	<b>Tiempo real</b>
Planteamiento escenarios	2h	2h
Pruebas de escenarios	5h	5h
<b>Total</b>	<b>7h</b>	<b>7h</b>

Tabla 1-4 - Tarea 4: Pruebas y validación.

<b>Subtarea</b>	<b>Tiempo estimado</b>	<b>Tiempo real</b>
Memoria del trabajo	90h	90h
<b>Total</b>	<b>90h</b>	<b>90h</b>

Tabla 1-5 - Tarea 5: Documentación.

En total, el trabajo ha requerido de 216h.



## 2 ESTADO DEL ARTE

---

Este capítulo tiene como objetivo informar sobre el estado actual de todas tecnologías que vamos a utilizar en este proyecto. Es importante explicar su funcionamiento y así comprender cómo funciona este proyecto para entender su calado.

En un primer lugar, definiremos de manera general IOT, para poder después meternos de lleno en la plataforma FIWARE. También introduciremos al lector en encriptación para que pueda entender el funcionamiento del trabajo, y terminaremos nombrando las diferentes herramientas que también nos han sido necesarias para desarrollar el trabajo.

### 2.1 Internet Of Things

El progreso en la tecnología y más concretamente en Internet es la fuente de lo que hoy conocemos como *Internet Of Things*. Las capacidades de las redes de telecomunicaciones permiten una vasta cantidad de elementos interconectados y la velocidad que permite la llegada del 5G a las redes móviles lo ha terminado de acelerar.

En concepto, se trata de la posibilidad de interconectar todo elemento controlable usando la red de internet para que el protocolo HTTP permita que los usuarios puedan manejar cualquiera de estos elementos, o estos mismos puedan automatizarse para que lo hagan entre ellos.

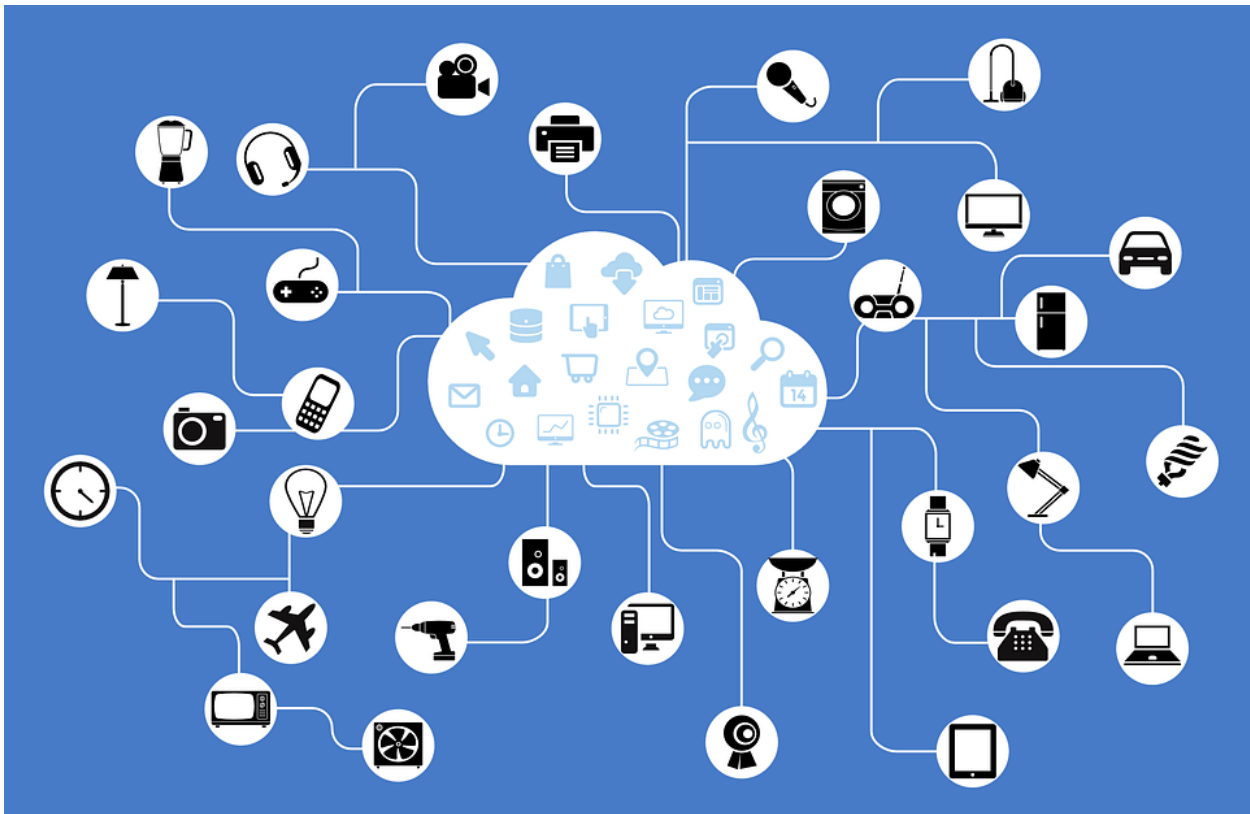


Ilustración 5- Internet Of Things

Lo que permite IOT es poner el mundo físico y la información en connivencia, para así ayudar al avance humano. Los sensores cobran un importante rol, haciendo de puente entre estos espacios. Extraen datos del medio para generar información y poder crear un contexto de ella. Así se consigue monitorizar el medio y dar respuestas incluso, si son necesarias [1].

Las comunicaciones podrán ser *human-human*, *human-thing* o *thing-thing*, aunque las “cosas” serán las principales creadoras de tráfico. Un ejemplo fácil y sencillo podría ser un sensor en un jardín, que recopila la temperatura en él, y te manda la información a tu teléfono móvil. Esto lo podemos extender hasta sensores en autopistas que detecten si la ocupación de la vía es demasiado densa u otros en plantas solares que detecten cuál es el momento y el ángulo de mejor captación solar. El límite de esta tecnología es infinito y nos encontramos en un momento en el que está en constante desarrollo.

En nuestro caso, vamos a plantear IOT en un escenario sanitario, en el que, por ejemplo, los sensores estén constantemente recopilando datos de los pacientes, para poder los médicos acceder a ellos cuando les haga falta saber algo del sujeto.

## 2.2 FIWARE

A pesar de que IOT sea un concepto avanzado, no existe un estándar normalizado sobre él. FIWARE es la apuesta de la UE para fomentar el despliegue de las plataformas IOT. Para ello usará el protocolo NGSI, que describiremos más adelante.



Ilustración 6 - Logotipo oficial de FIWARE

FIWARE es una plataforma de componentes de código abierto, con el fin de desarrollar “soluciones inteligentes” [2]. Surge de la colaboración público-privada entre la Comisión Europea y la industria europea en 2011. Alrededor de ella se ha creado un ecosistema de innovación constituido por desarrolladores de aplicaciones, proveedores de tecnología y por las entidades que demandan soluciones [3].

Aunque FIWARE es una plataforma genérica, está pensada y es especialmente útil para el concepto de ciudad inteligente, y más de 15 ciudades europeas han publicado ya datos abiertos y desarrollados prototipos con ella, al tiempo que varias ciudades españolas la han adoptado en sus sistemas [3].

Mediante las herramientas de gestión de operaciones de FIWARE (FIWARE Ops) y los *Generic Enablers* (Componentes) es posible desplegar y gestionar instancias FIWARE que permitan a los usuarios desplegar contextos según sus necesidades.

Un elemento esencial en la arquitectura FIWARE es la gestión del contexto. Para desarrollar aplicaciones o servicios inteligentes, es fundamental que éstos tengan información sobre lo que sucede en cada momento. FIWARE proporciona un mecanismo que permite generar, recopilar, publicar y consumir información de contexto de forma masiva y hacer uso de ella desde las aplicaciones, haciendo posible que reaccionen a lo que sucede [3].

La información de contexto se representa mediante valores asignados a atributos, que a su vez caracterizan a aquellas entidades que representan a la aplicación. Del almacenaje de esta información se encarga el componente Context Broker, cuya implementación de referencia es Orion, y que es capaz de gestionar esta información de contexto de manera masiva. Orion implementa el API estándar NGSI (Next Generation Services Interface) que permite a las aplicaciones preguntar sobre el contexto e incluso suscribirse a cambios del mismo, que se reciben a través de notificaciones. Para ello utiliza HTTP, variando con peticiones GET(solicitar dato), POST(actualizar entidad) o PATCH(actualización de valor de atributo) y la dirección url, para manejar los contextos.

La información de contexto puede provenir de muy diferentes fuentes: sistemas existentes, usuarios con una aplicación móvil, redes de sensores, etc. En el caso que concierne a este proyecto, la recibirá del componente IOT Agent, que ejerce de puente entre el sensor y el Context Broker.

Finalmente, cabe decir que utilizando la API NGSI es posible acceder a toda la información relevante con independencia de la infraestructura IoT, e incluso de servicios y aplicaciones. Además, es posible no sólo leer información procedente de sensores, sino incluso tener actuadores, simplemente cambiando un determinado atributo asociado a la entidad del actuador. Por tanto, podríamos hacer incluso una analogía entre SNMP y NGSI, de forma que este último podría considerarse el “SNMP de la Internet de las Cosas”.

A continuación, pasaremos a describir los dos elementos más importantes de nuestra plataforma basada en FIWARE.

### 2.2.1 Orion Context Broker

Orion Context Broker (OCB) es el *Generic Enabler* núcleo y elemento obligatorio de cualquier plataforma implementada según FIWARE. Permite el manejo y guardado de contextos de información de toda la red IOT. Para ello se apoya de la base de datos de MongoDB para el almacenamiento de datos.

Utiliza una interfaz NGSI con la que el cliente puede realizar diversas operaciones: Almacenar datos, Consultar datos o Suscribirse a datos. Actualmente la primera versión se encuentra obsoleta, luego para configurar nuestros servicios FIWARE especificaremos NGSI-v2.

La información de contextos se encuentra representada en entidades. Éstas pueden tener múltiples atributos y valores. Para una mayor simplicidad en la representación de atributos se suele apoyar en JSON.

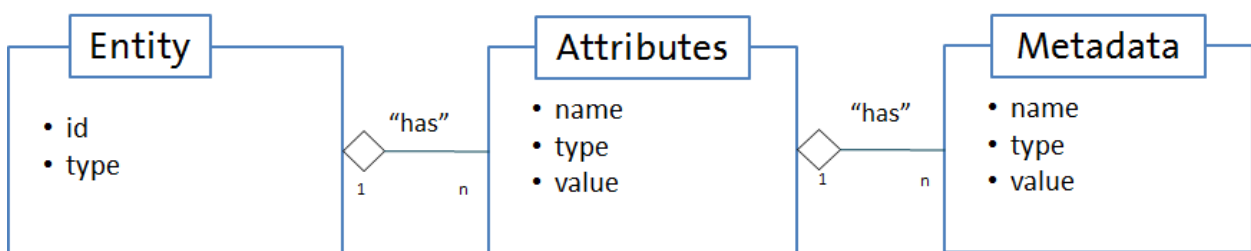


Ilustración 7 - Esquema de contextos de información basados en entidades

Orion está implementado en C++ y basado en la API REST NGSI-v2. Actualmente, se encuentra en su versión 3.1.0. Escucha en el puerto por defecto 1026. Su fichero de configuración se encuentra en `etc/default/contextBroker`.

### 2.2.2 IOT Agent-JSON

Se trata de un *Generic Enabler* que nos ofrece la posibilidad de obtener un puente entre nuestros sensores y el Context Broker. Más concretamente, IOTA nos facilita la tarea de establecer una interfaz NGSI en los sensores, puesto que estos se conectan mediante otros protocolos como pueden ser MQTT, LWM2M, LoRaWan o una simple muestra HTTP [4].

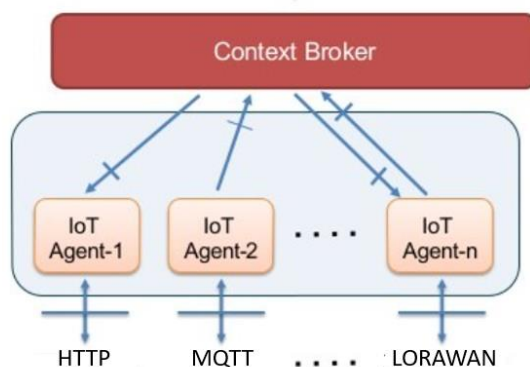


Ilustración 8-Esquema de interconexión del IOT Agent

IOTA está dividido conceptualmente en dos puertos de entrada y salida llamados polos, uno al “norte” que conecta con el Orion CB, y otro al “sur” que es el encargado de comunicarse con el sensor. Las peticiones que vamos a estudiar son las *Northbound*, en sentido Norte, que quiere decir desde el sensor al Orion.

En concreto, el IOT Agent elegido se trata del basado en JSON. Los IOTA-JSON gestionan un único mensaje HTTP. Éste incluye la información a manejar, el payload o carga, y su contenido suele ser JSON, ya sea para almacenar (PUT) o consultar (GET). El proyecto se encuentra desarrollado en Javacript y se encuentra en su versión 1.18.0.

## 2.3 Encriptación

La criptografía siempre ha sido de vital importancia en la historia de la humanidad. Existen estudios que indican que los griegos y los romanos realizaban mecanismos de sustitución, intercambio de lugar de las letras del abecedario, como el cifrado César para evitar que sus mensajes fueran captados por su enemigo [5]. Y de ahí se fue lentamente desarrollando hasta el siglo XX, con las Guerras Mundiales, en la que los países implicados se encargaron de mejorar sus apartados científicos, desarrollando las primeras máquinas que permitieron implementar métodos criptográficos tales como Enigma.

### 2.3.1 Criptografía Moderna: Cifrado simétrico y asimétrico

Con el desarrollo de los primeros ordenadores, se empezó a pensar en métodos mucho más elaborados de encriptación, puesto que con ellos también sería más fácil la descryptación. Se implementó el concepto de clave como contraseña para transmutar los mensajes y así hacerlos ilegibles. Hoy en día, existen dos métodos generales para llevarlas a cabo, simétrico y asimétrico.

El método simétrico consta de una clave secreta de la que es conocedor tanto el emisor como el receptor. Con ella encriptamos un mensaje y le asegura confidencialidad e integridad a la transmisión, hasta que un método análogo inverso la descifre en el receptor. Ejemplos de éste pueden ser DES, o su evolución, AES.

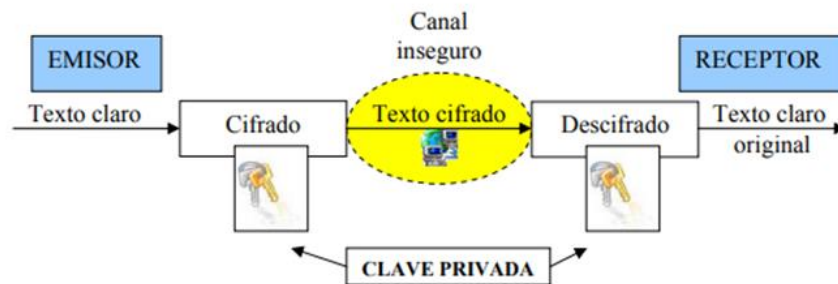


Ilustración 9 - Esquema de funcionamiento de cifrado simétrico

El método asimétrico es más completo puesto que se compondría de dos claves, una secreta y una pública, para cada usuario. Éste asegura más confidencialidad en la clave, puesto que no hay paso de claves privadas a otros usuarios, más que la pública de cada uno. Aquí el emisor cifra el mensaje con la clave pública del receptor, y ésta está hecha a conciencia para que sea descifrada por la clave privada del receptor. Con este método aseguramos también la autenticidad de los usuarios en el intercambio de información.

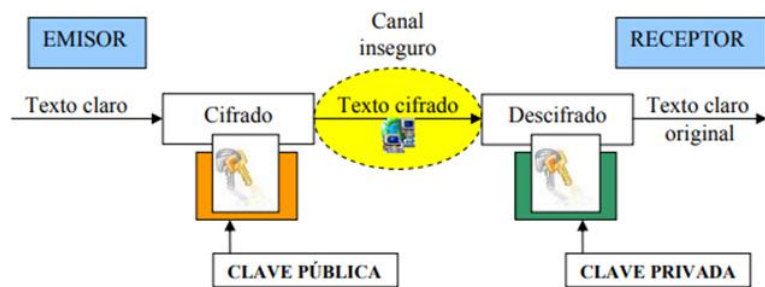


Ilustración 10 - Esquema de funcionamiento de cifrado asimétrico

## 2.3.2 AES

En enero de 1997, el Instituto Nacional de Estándares y Tecnología (NIST) anunció un concurso para desarrollar el AES, un nuevo sistema de encriptación mucho más seguro que el anterior DES-que debido al avance en la computación quedó obsoleto puesto que descifrarlo había pasado a ser una tarea fácil-, e hizo una convocatoria formal de algoritmos el 12 de septiembre 1997. Después de revisar los resultados de esta investigación preliminar, los algoritmos MARS, RC6™, Rijndael, Serpent y Twofish fueron seleccionados como finalistas. Y tras un intenso análisis sobre los finalistas, NIST decidió proponer a Rijndael como el cifrado avanzado estándar (Advanced Encryption Standard).

### 2.3.2.1 Funcionamiento

AES implementa una solución de clave simétrica, en la que tanto el emisor del mensaje como el receptor comparten una clave común, que sirve para encriptar y descifrar el mensaje.

El algoritmo de Rijndael está hecho con la intención de que, si se intenta descifrar una clave por una máquina, esta tarea sea tan difícil que requiera de muchos años para ello. Esto se debe a su clave de 128 bits (estamos hablando de  $3.4028237^{38}$  posibilidades), aunque actualmente ya se emplean claves de 192 y 256 bits.

Este algoritmo consta de 4 operaciones principales llamadas *AddRoundKey*, *SubBytes*, *ShiftRows* y *MixColumns*, y se dividen en rondas, según el tamaño de la clave, que se divide cada 4 bits. Consisten, correspondientemente, en: añade la parte de la clave de la ronda, sustituye bytes con un bloque predeterminado, cambia el orden de las filas y mezcla el orden de las columnas [6].

En la siguiente tabla vemos exactamente el proceso, junto con el de la descifricación, que es totalmente análogo e inverso:

AES Encryption	AES Decryption
AddRoundKey	InvAddRoundKey
-for round=1 to Nr-1	-for round=1 to Nr-1
SubBytes	InvShiftRows
ShiftRows	InvSubBytes
MixColumns	InvAddRoundKey
AddRoundKey	InvMixColumns
-end for	-end for
SubBytes	InvShiftRows
ShiftRows	InvSubBytes
AddRoundKey	InvAddRoundKey

Tabla 3.1.1.2 - Pasos para la Encriptación y  
Decriptación mediante AES

Como vemos es un algoritmo muy difícil de ser descryptado sin la contraseña, al ser muy elaborado, y la contraseña a su vez, es también muy difícil de conseguir.

### 2.3.3 HTTPS

La búsqueda de la comunicación segura llevó a Netscape Communications, empresa del mítico navegador Netscape, a desarrollar el protocolo SSL en 1992. En teoría, se trata de un protocolo que ponemos sobre HTTP para que pueda asegurar la encriptación del canal, y para ello hace uso del puerto 443, en vez del 80 de HTTP. El funcionamiento de esta nueva capa tiene como objetivo crear un túnel seguro sobre el que implementar el flujo normal de la información.

En marzo de 1995, su versión 2 se incorporó en el navegador Netscape Navigator 1.1, lo cuál resultó en la primera vez que se utilizaba un protocolo de cifrado en un navegador, algo que hoy en día se considera fundamental.

El protocolo avanzó en su versión 3 para ser base de lo que iba a desarrollar en 1999 la IETF, TLS 1.0, siendo actualmente el estándar de la mayoría de los navegadores TLS 1.2.

#### 2.3.3.1 Funcionamiento

HTTPS encripta información durante una transmisión en Internet. Todo empieza cuando se instala un certificado digital SSL/TLS en un servidor y se asocia con su dominio. Un certificado digital es un documento que contiene diversos datos, entre ellos el nombre de un usuario y su clave pública, y que es firmado por una Autoridad de Certificación (AC). Como emisor y receptor confían en dicha AC, el servidor podrá autenticarse ante el usuario.

Los sistemas de clave pública, también llamados de cifrado asimétrico, emplean dos claves para cifrado y descifrado de la información. Ambos interlocutores poseen una clave pública y una clave privada, generando su clave pública en base a la clave privada [7]. Así, cualquier mensaje que se encripte con la clave pública de un usuario, solo se puede descryptar con la clave privada de ese mismo usuario.

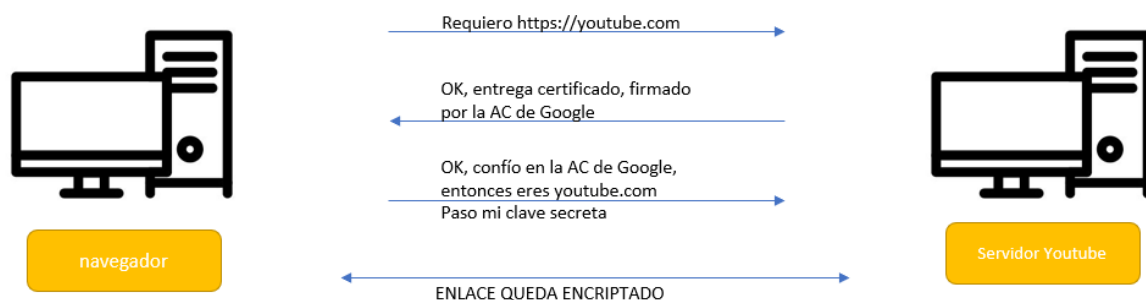


Ilustración 11 - Esquema de ejemplo de uso de HTTPS

En la Ilustración 11 encontramos un ejemplo de cómo funciona HTTPS. Al hacer una petición segura, el servidor devolverá primero el certificado. Ya que tenemos la clave pública en este caso de Google, es fácil comprobar que se trata de una AC en la que confiamos. Entonces, el navegador pasa su clave secreta creada especialmente para esta conexión, cifrada con la clave pública que le ha pasado Youtube. Tras el intercambio

de claves, queda implementado el enlace encriptado.

Es en la entrega de certificados donde se implementa el mensaje Client Hello de TLS, que en un ejemplo del apartado Cipher Suite se especifican los siguientes elementos de un enlace encriptado HTTPS: TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256.

Se dividiría en ECDHE, como algoritmo de intercambio de claves. Otros: DH, DHE, ECDH, ECHE, RSA, PKCS, DSA, CK.

RSA, como algoritmo de firma digital. Otros: ECDSA, DSS.

AES\_128, como algoritmo de cifrado de datos. Otros: RC2, RC4, RC5, DES, 3DES, IDEA.

GCM, como modo de encriptación. Otros: CBC, ECB, PCBC, CFB, OFB, CTR.

SHA256, como algoritmo de hash. Otros: SHA, SHA384, AEAD, HMAC, MD2, MD4, MD5, MDC2.

También, dependiendo del algoritmo de intercambio de claves, pueden añadir el tamaño de curva elíptica: P256 y P384.

## 2.4 Herramientas utilizadas

### 2.4.1 Máquina Virtual e imagen Ubuntu 20.04 LTS

La máquina virtual por utilizar se trata de una Vmplayer Workstation y en ella vamos a instalar, para llevar a cabo nuestro trabajo, una imagen de Ubuntu 20. Ubuntu está basada en Linux y es la distribución principal en desarrollo de software. Con ésta nos será fácil tanto configurar los programas como manejar la conexión entre ellos.

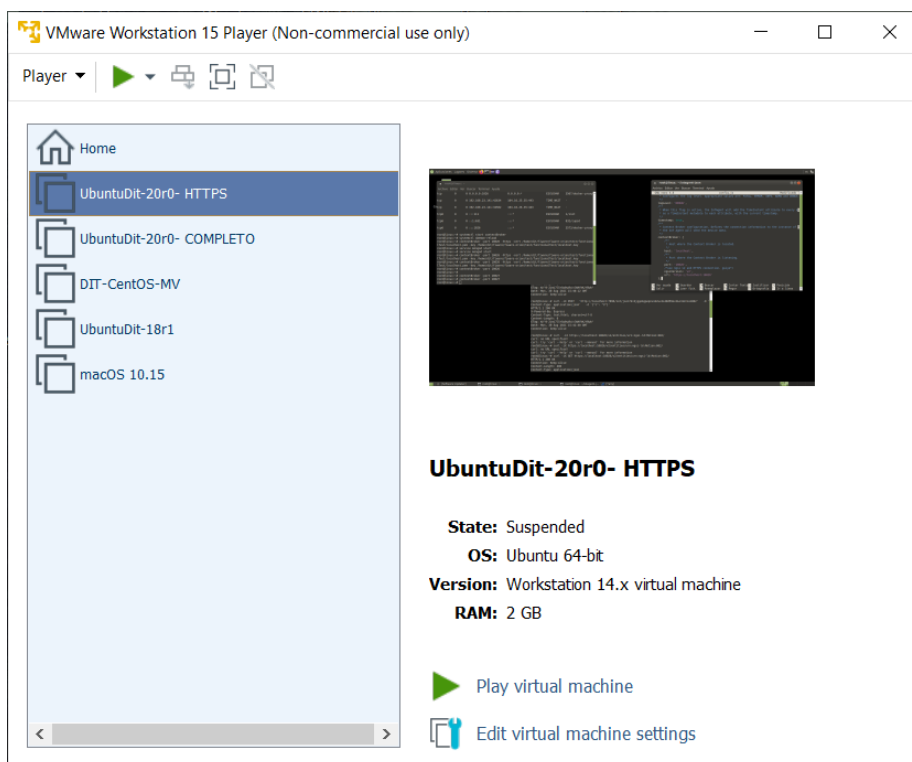


Ilustración 12 -Vmplayer Workstation corriendo máquina virtual Ubuntu



## 2.4.2 MongoDB

Para el almacenamiento de los contextos, Orion requiere de la base de datos de MongoDB. Ésta tiene una característica que la hace muy dinámica, y es que guarda los datos en ficheros tipo JSON [8], que es como nosotros transmitimos y guardamos nuestra información. Por ello, es una de las bases de datos más empleadas en el mundo.



Ilustración 13 - Fichero JSON que gestiona MongoDB

## 2.4.3 cUrl

Se trata de una librería de código abierto para Linux, que permite la transferencia de datos y archivos por internet. *cUrl* es usada en línea de comandos o scripts para hacer estas peticiones HTTP, FTP, etc. Es el motor de transferencia por internet de todo tipo de dispositivos, con miles de millones de instalaciones [9].



Ilustración 14 - Logotipo oficial de la librería cUrl

## 2.4.4 Wireshark

Wireshark es el más extendido analizador de paquetes. Permite escanear lo que pasa en tu red, en cualquier capa de ella, según pasa por tu ordenador. Corre en cualquier plataforma como Windows, Linux, Solaris, FreeBSD, NetBSD, etc. Con su capacidad de capturar paquetes podemos ver qué se intercambia en cualquier

momento, para así analizar los paquetes, y aprovecharnos de ello para nuestros intereses [10].

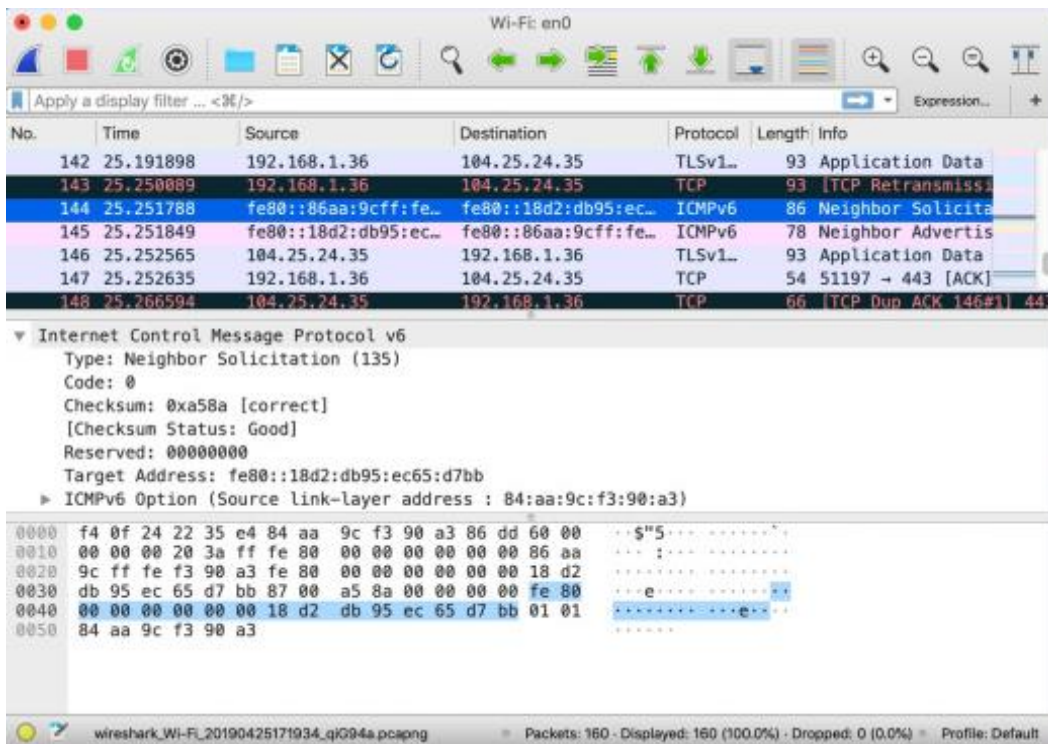


Ilustración 15 - Ejemplo de captura de paquetes del programa Wireshark

## 2.4.5 openssl

Es una librería muy útil con herramientas para trabajar en Linux con SSL y TLS. Es un proyecto de código abierto y muy extendido, y nos ayudará a crear certificados y claves privadas. Además, también tiene funciones de criptografía [11]. Su función principal es generar certificados autofirmados.



Ilustración 16 - Logotipo oficial de la librería openssl

# 3 RESULTADOS

Para la encriptación vamos a plantear dos escenarios. Estos dos los vamos a estudiar para terminar desarrollando uno de ellos al final del capítulo, y así ponerlo a prueba. Para ello, se mostrará primero la ejecución de la plataforma. Luego se describirá el mecanismo de encriptación AES, para discutir su viabilidad en la plataforma. Posteriormente haremos lo mismo con HTTPS. Por último, se planteará el escenario elegido para implementar y se analizará mediante Wireshark que la encriptación se ha llevado correctamente a cabo. Así, terminaremos dando una solución a la problemática de la confidencialidad de los datos en un entorno real de intercambio de información en una red IOT. Además, se añadirá un apartado para la comparación de uso de recursos que conlleva esta modificación.

## 3.1 Ejecución de componentes de la plataforma basada en FIWARE

Para este proyecto se va a crear una plataforma IOT en la que podamos simular una serie de sensores, y que, puenteados con un IOT Agent, puedan agregar su información de contexto a un Orion Context Broker.

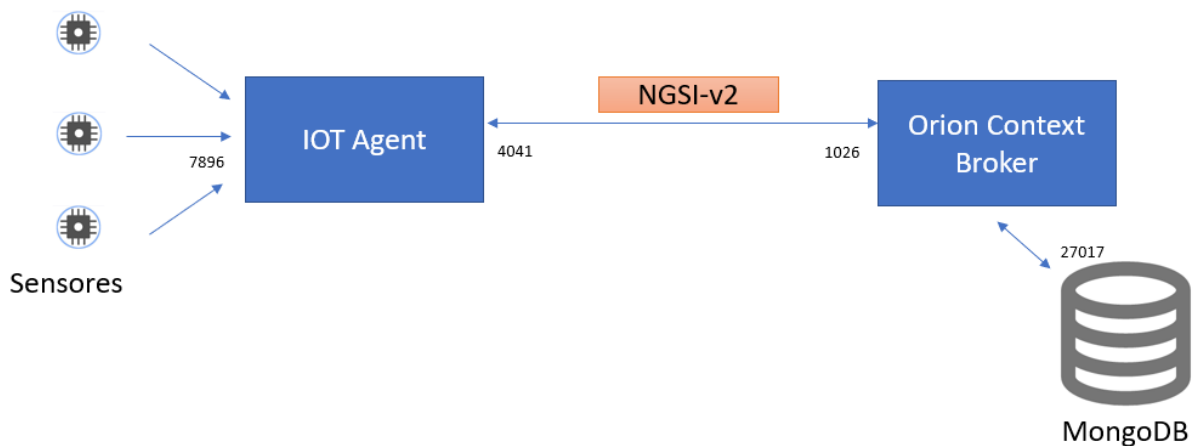


Ilustración 17 - Esquema de la plataforma IOT basada en FIWARE que se va a implementar

Para iniciar la plataforma, primero instalaremos OCB, con su herramienta necesaria MongoDB, y las pondremos a funcionar. Posteriormente, procederemos a hacer lo mismo con IOTA. Los sensores serán simulados por peticiones HTTP realizadas por la herramienta cUrl.

### 3.1.1 Instalación y ejecución de Orion

Para instalarlo encontraremos en su página oficial los pasos necesarios [12]. Esta instalación será desarrollada en el Anexo A. Una vez instalado, tan solo tendremos que dar de alta su servicio y arrancar el programa con los comandos:

```
>> systemctl daemon-reload
>> systemctl start contextBroker
>> contextBroker -port 10027
```

Para manejar OCB, lo primero que tenemos que hacer es crear una entidad, con su tipo, y con todos los atributos que se requieran, que tendrán un tipo y un valor. Todo ello irá en una única cadena JSON, en el *payload* de la petición. Nos vamos a ayudar de una entidad de ejemplo que llamaremos “Ejemplo5” de tipo “Ejemplo” para mostrar su uso:

```
>> curl localhost:10027/v2/entities -s -S --header 'Content-Type:
application/json' -X POST -d @- <<EOF
{
  "id": "Ejemplo5",
  "type": "Ejemplo",
  "atributo1": {
    "value": ,
    "type": "Number"
  },
}
EOF
```

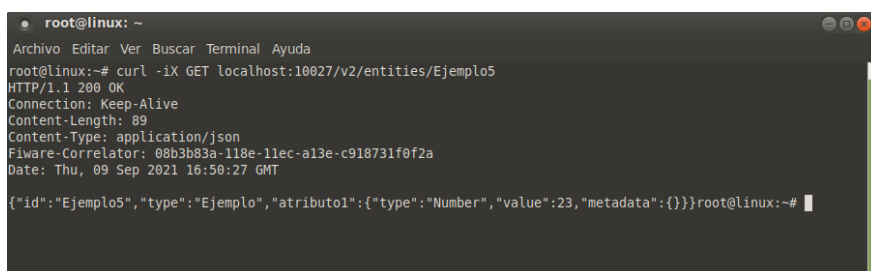
A esta entidad se le pueden añadir, o exactamente actualizar, valores:

```
>> curl localhost:10027/v2/entities/Ejemplo5/attrs/atributo1/value -s -S
--header 'Content-Type: text/plain' -X PUT -d 10
```

Para obtener una entidad puedes o demandarla entera o un valor concreto:

```
>>curl localhost:10027/v2/entities/Ejemplo5 -s -S --header 'Accept:
application/json' | python -mjson.tool
>>curl localhost:10027/v2/entities/Ejemplo5/attrs/atributo1/value -s -S -
-header 'Accept: text/plain' | python -mjson.tool
```

Al final de la instalación y las pruebas de funcionamiento podemos comprobar que ha funcionado con éxito y se ha creado el contexto ‘Ejemplo5’.



```
root@linux:~
Archivo Editar Ver Buscar Terminal Ayuda
root@linux:~# curl -iX GET localhost:10027/v2/entities/Ejemplo5
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Length: 89
Content-Type: application/json
Fiware-Correlator: 08b3b83a-118e-11ec-a13e-c918731f0f2a
Date: Thu, 09 Sep 2021 16:50:27 GMT
{"id": "Ejemplo5", "type": "Ejemplo", "atributo1": {"type": "Number", "value": 23, "metadata": {}}}root@linux:~#
```

Ilustración 18 - Entidad 'Ejemplo5' creada

### 3.1.1.1 Fallos en el uso de Orion y cómo contrarrestarlos

Los procesos de Orion Context Broker tienden a no cerrarse correctamente. Se debe mirar en su *log* para comprender porqué tarda a veces en arrancar, ejecutando `cat tmp/contextBroker.log`. Muchas veces obtenemos errores que impiden su funcionamiento, como FATAL ERROR RESTIFY. En cualquier caso, lo mejor es buscar sus procesos con el comando `ps ax | grep contextBroker` y hacer un `kill` a sus pid. Otras veces lo necesario será encontrar los procesos que bloquean el puerto 1026 que utiliza Orion por defecto, `ps ax | grep 1026` ó `netstat -tanp`. Además, también sería útil eliminar el archivo donde se guarda el pid del contextBroker, suele dar fallos al no haberse eliminado correctamente.

Como solución alternativa y más rápida, se puede especificar en el arranque que contextBroker empiece en un puerto libre diferente al 1026; por ejemplo, 10027.

### 3.1.2 Instalación y ejecución de IOT Agent

Para instalarlo simplemente descargamos su paquete y realizamos una instalación mediante Node-JS. El arranque se dará ejecutando simplemente su archivo principal en la carpeta `bin`:

```
>> git clone https://github.com/telefonicaid/iotagent-json.git
>> npm install
>> bin/iotagent-json
```

Su fichero de configuración será `config.js`, y es necesario, antes de ponerse a trabajar con IOTA, modificar la versión NGSI, puesto que por defecto viene la v1. Por tanto, dentro del apartado `config.iota`, encontraremos `contextBroker`. Ahí añadiremos la siguiente clausula:

```
...
config.iota = {
...
  contextBroker = {
...
    ngsiVersion: 'v2'
...

```

`config.js`

Para hacer uso de IOTA, tenemos que registrar el sensor. Para ello, primero hay que mandarle una petición de creación en el contexto de Orion de un *service*, que sería el servicio o propósito al que está destinado, y luego la de un *device*. Pero no lo enviaremos directamente a Orion, si no al puerto de IOTA destinado a la configuración 4041, y así éste se autoconfigura mediante NGSI-v2 con el otro. Lo haremos en dos peticiones HTTP de ejemplo, para así crear un sensor `'motion002'`.

```
>> curl -iX POST 'http://localhost:4041/iot/services' -H 'Content-Type: application/json' -H 'fiware-service: openiot' -H 'fiware-servicepath: /' -d '{
"services": [
```

```

{
  "apikey":      "4jggokgpepnvsb2uv4s40d59ov2",
  "cbroker":    "http://orion:1026",
  "entity_type": "Thing",
  "resource":   "/iot/json"
}
]
}'
>> curl -iX POST      'http://localhost:4041/iot/devices'      -H 'Content-
Type: application/json'      -H 'fiware-service: openiot'      -H 'fiware-
servicepath: /'      -d '{
  "devices": [
    {
      "device_id":    "motion002",
      "entity_name":  "urn:ngsi-ld:Motion:002",
      "entity_type":  "Motion",
      "timezone":     "Europe/Madrid",
      "attributes": [
        { "object_id": "c", "name": "count", "type": "Integer" }
      ],
      "static_attributes": [
        { "name": "refStore", "type": "Relationship", "value": "urn:ngsi-
ld:Store:002" }
      ]
    }
  ]
}'

```

Se ha remarcado el atributo `apikey` de `service`, ya que será importante para la actualización de datos por parte del sensor.

Estas peticiones, mediante la herramienta `curl`, constan de: una serie de opciones del comando, modalidad de envío HTTP, dirección URL de envío, especificación del tipo del contenido que enviamos (JSON en nuestro caso), `fiware-service` y `fiware-servicepath` (clausulas obligatorias), y por último la carga o *payload*. Esta última será la que defina nuestro servicio, sensor y nuestro contexto en sí.

El segundo comando lo que contiene en la carga es la creación de la entidad que va a manejar el sensor cuyo nombre será el del valor `entity_name`. Esto es importante para saber a qué entidad hay que consultar después los datos.

Una vez realizado el registro del `device`, ya podemos enviar datos desde nuestro sensor. Lo podemos emular con la herramienta `cUrl`, enviando nuestra propia petición al puerto que tiene preparado IOTA para recibir datos mediante HTTP, 7896, especificando nuestro *service* y nuestro *device*. Dentro del *payload* meteremos una cadena JSON con el id del atributo a actualizar y su valor.

```
>> curl -iX POST 'http://localhost:7896/iot/json?k=4jggokgpepnvsb2uv4s40d59ov2&i=motion002' -H 'Content-Type: application/json' -d '{"c": "2"}'
```

Al final de todas estas pruebas, realizamos la comprobación de que la entidad del sensor 'motion002' se haya creado con éxito, y que la actualización de su valor funcione. Haremos una consulta mediante cUrl y realizando una petición al contexto sobre la entidad en cuestión.

```
root@linux: ~
Archivo Editar Ver Buscar Terminal Ayuda
root@linux:~# curl -iX POST 'http://localhost:7896/iot/json?k=4jggokgpepnvsb2uv4s40d59ov2&i=motion002' -H 'Content-Type: application/json' -d '{"c": "2"}'
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 0
ETag: W/"0-2jmj7l5rSw0yVb/vlWAYkK/YBwk"
Date: Thu, 09 Sep 2021 17:01:14 GMT
Connection: keep-alive

root@linux:~# curl -iX GET localhost:10027/v2/entities/urn:ngsi-ld:Motion:002
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Length: 400
Content-Type: application/json
Fiware-Correlator: 8dc27fce-118f-11ec-a13e-c918731f0f2a
Date: Thu, 09 Sep 2021 17:01:19 GMT

{"id": "urn:ngsi-ld:Motion:002", "type": "Motion", "TimeInstant": {"type": "DateTime", "value": "2021-09-09T17:01:14.881Z"}, "metadata": {}, "count": {"type": "Integer", "value": "2", "metadata": {"TimeInstant": {"type": "DateTime", "value": "2021-09-09T17:01:14.881Z"}}}, "refStore": {"type": "Relationship", "value": "urn:ngsi-ld:Store:002", "metadata": {"TimeInstant": {"type": "DateTime", "value": "2021-09-09T17:01:14.881Z"}}}
root@linux:~#
```

Ilustración 19 - Comprobación del arranque y funcionamiento de IOT AGENT

Con esto finalizaríamos la puesta a punto de la plataforma al completo, y quedaría arrancada y preparada para la realización de las pruebas de encriptación que se van a realizar en los siguientes apartados.

## 3.2 Análisis teórico de los mecanismos de encriptación

### 3.2.1 Análisis de la aplicación de AES a la plataforma IOT

En el escenario que nos concierne, se trataría de encriptar el valor con AES. El resto de la transmisión iría en texto plano, pero no supondría un problema puesto que no ponemos los datos del paciente en peligro.

Se podría haber pensado en la opción de encriptar todo el canal con AES, pero hay un inconveniente mayor. Implementar nuestro escenario es sencillo de aplicar. Una vez analizado el código de IOTA-JSON, podemos encontrarnos nuestro *payload* para poder así llevar a cabo la encriptación del valor. Sin embargo, para encriptar todo el canal tendríamos que modificar toda la parte encargada de la transmisión de datos. Lo cual, además de ser demasiado laborioso para desarrollarlo por uno mismo, ya existe una solución más fácilmente implementable que se explicará en el escenario 2.

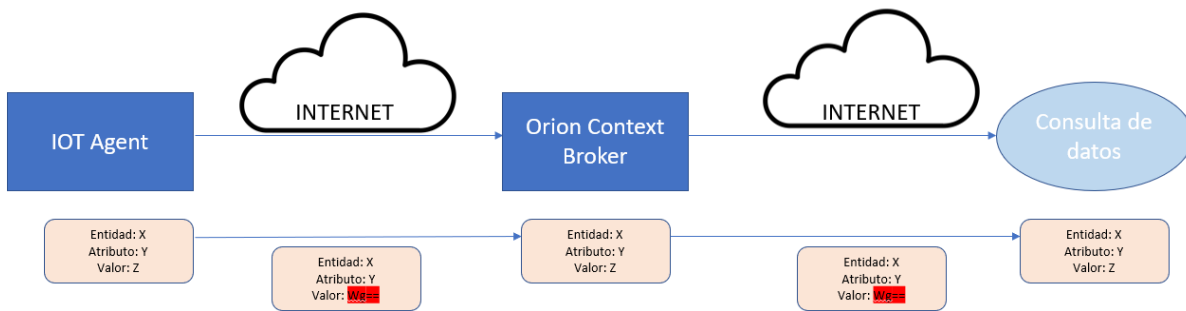


Ilustración 20-Esquema del escenario 1

Uno de los inconvenientes del primer escenario se trataría de dónde podríamos transmitir la clave al Orion. Este es un paso obligatorio ya que Orion necesita disponer de los valores para hacer comparativas, y así poder poner alarmas a los suscriptores de los datos. Por tanto, propongo que pueda viajar al registrar el *device* en la puesta a punto del IOTA, pero queda definir el cómo.

Esta encriptación ofrece mayor confidencialidad de datos, ya que no sería un protocolo de encriptación que esté principalmente distribuido en otras conexiones, luego debería resultar preferible. Aunque uno de los problemas de éste es que también tiene un mayor coste. Sobretodo de operabilidad, intentar implantarlo podría ser muy engorroso. A pesar de las diferentes librerías que podrían llevar a cabo ese trabajo, se tendría que investigar y modificar IOTA y Orion para que se implantasen.

### 3.2.2 Análisis de la aplicación de HTTPS en la plataforma IOT

En este escenario, una vez analizada la teoría de TLS, podemos comprobar que también se emplea AES como encriptación de datos en la conexión. Sin embargo, es HTTPS la que se encarga de configurarlo. Por tanto, podemos tratarlo como un modo de encriptación diferente a ojos del desarrollador, porque no tenemos que configurar ni implementar nada parecido.

El objetivo ahora es modificar en la plataforma FIWARE que todas las conexiones NGSI-v2, que van sobre HTTP, pasen a hacerse de modo seguro bajo HTTPS. En principio solo habrá que modificar las *request* para que actúen como tal, luego parece una modificación simple a primera vista. Gracias a ello, conseguimos el cifrado del canal completo, sin tener que manejar directamente el valor, que pasaría a tener un rol secundario en nuestra encriptación.

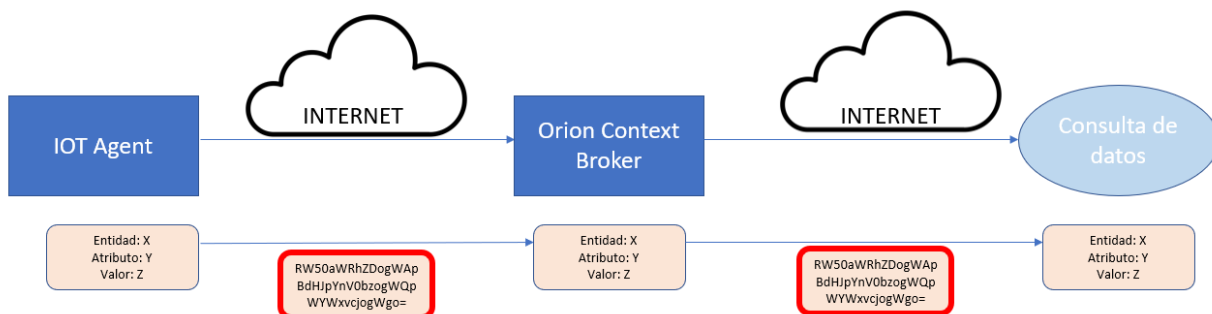


Ilustración 21- Esquema de descripción del escenario 2



Desgraciadamente, al ser HTTPS un protocolo muy implementado en todo el mundo, esto lo convierte en objetivo recurrente de hackers maliciosos, y ya existen técnicas, como ejecutar *Man-in-the-Middle*, y poder descifrar el envío de datos [13].

Por otra parte, diseñar este escenario nos será mucho más simple y eficiente, es suficientemente seguro y, además, poder encriptar el canal completo da más sensación de seguridad. Por tanto, vamos a desarrollar experimentalmente éste en el siguiente apartado.

### 3.3 Implementación del mecanismo de encriptación HTTPS en la Plataforma IOT

Una vez elegida la encriptación que vamos a llevar a cabo, la vamos a simular en nuestra máquina virtual de manera local. Por tanto, emularemos el sensor y las direcciones ip de IOTA y OCB.

#### 3.3.1 Modificación de Orion

Para llevar a cabo la securización de Orion nos guiamos de los pasos especificados en [14]. Cabe decir que modificar Orion para que se ejecute con HTTPS incapacitaría su uso con HTTP.

En principio, lo único necesario para implementar Orion securizado sería ejecutar un comando, pasándole el archivo del certificado digital que implementará Orion y su clave secreta:

```
>> contextBroker -https -cert localhost.pem -key localhost.key
```

Para conseguir esto encontramos varios problemas. Para empezar que, como se comentó en el apartado 3.1.1.1 sobre la puesta a punto de Orion, el programa tiende a no cerrarse bien y, por tanto, resulta complicado llevar a cabo el arranque del programa de nuevo con https. Una forma alternativa para evitar el bloqueo del puerto 1026 es especificarle otro puerto libre de uso añadiendo la clausula `-port`:

```
>> contextBroker -port 10026 -https -cert localhost.pem -key localhost.key
```

Por ultimo, y lo más importante para hacer funcionar HTTPS, es el tema del certificado digital, conseguir `localhost.pem` y `localhost.key`. Se ha probado a crear un propio certificado y clave con el programa de seguridad `openssl`. Sin embargo, OCB no las acepta con la excusa de que considera las resultantes demasiado grandes en tamaño.

Los propios desarrolladores de Orion dan para esto un script de creación de una clave y contraseña, que se encuentra en el paquete del proyecto, en `test/functionalTest/httpsPrepare.sh`. Al ejecutarlo, da un certificado y clave de ejemplo que podremos usar para poner en marcha Orion securizado.

El problema es que éste se trata de un certificado autofirmado, y por tanto no fiable para cuando hagamos nuestras conexiones “`curl https://localhost:1026/...`”, lo cual no nos permite avanzar. Si tuviéramos una ip fija y un dominio asignado podríamos asociarnos a una AC fiable, como la de Google o muchas otras existentes, y así conseguiríamos la integridad necesaria para llevar a cabo las peticiones a nuestro OCB.

Para solucionar este problema, podríamos forzar que la herramienta `cUrl` confiara en estas conexiones, añadiendo la opción `-k`. Sin embargo, nos topamos con que en nuestro escenario no va a ser `cUrl` quien envíe los datos a Orion, sino IOTA, y éste tiene su propio mecanismo ajeno a `cUrl` para el envío de datos.

Por tanto, vamos a optar por la solución de meter el certificado de Orion `localhost.pem` en los certificados en los que confía nuestra AC raíz, la de nuestro ordenador en el que estamos simulando. Para ello he alojado en `/usr/local/share/ca-certificates` el certificado y a continuación he ejecutado el

comando “update-ca-certificates”.

### 3.3.2 Modificación de IOTA

La modificación de IOTA será más sencilla y dará menos problemas. Y es que lo único que tendremos que hacer especificar en el fichero de configuración que la dirección a enviar los datos es HTTPS. Por tanto, añadiremos una clausula `url`.

```
...  
config.iota = {  
...  
  contextBroker = {  
...  
    url: 'https://localhost:10026'  
...  
}
```

config.js

Es importante que, al hacer cualquier modificación en `config.js`, reinstalemos IOTA antes de volver a arancarlo con “`npm install`”.

### 3.3.3 Demostración de uso de la encriptación

Para probar experimentalmente nuestros resultados, vamos a simular en tres terminales Linux a Orion CB, IOTA y el sensor, respectivamente. Este apartado tiene como objetivo analizar que la plataforma no se encuentra encriptada y que funciona la encriptación pensada. Así, primero vamos a arrancar la plataforma en modo normal, para pasar luego a implementar HTTPS.

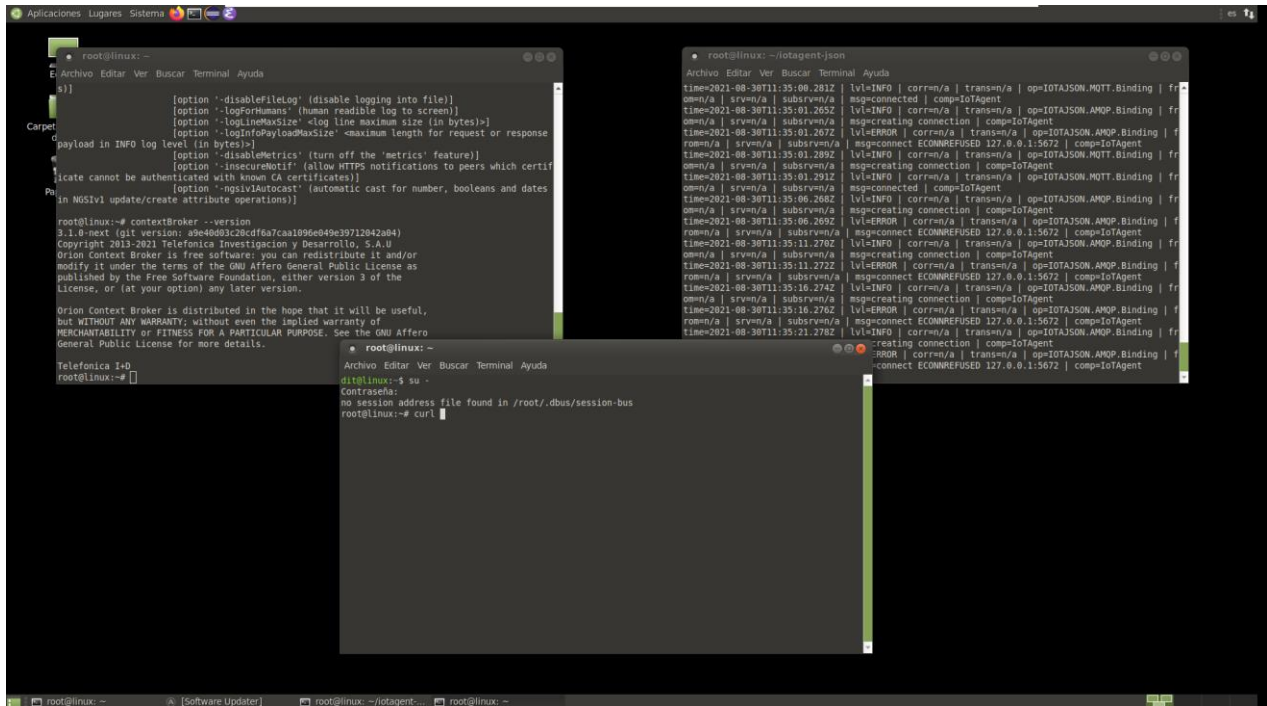


Ilustración 22- Simulación del escenario

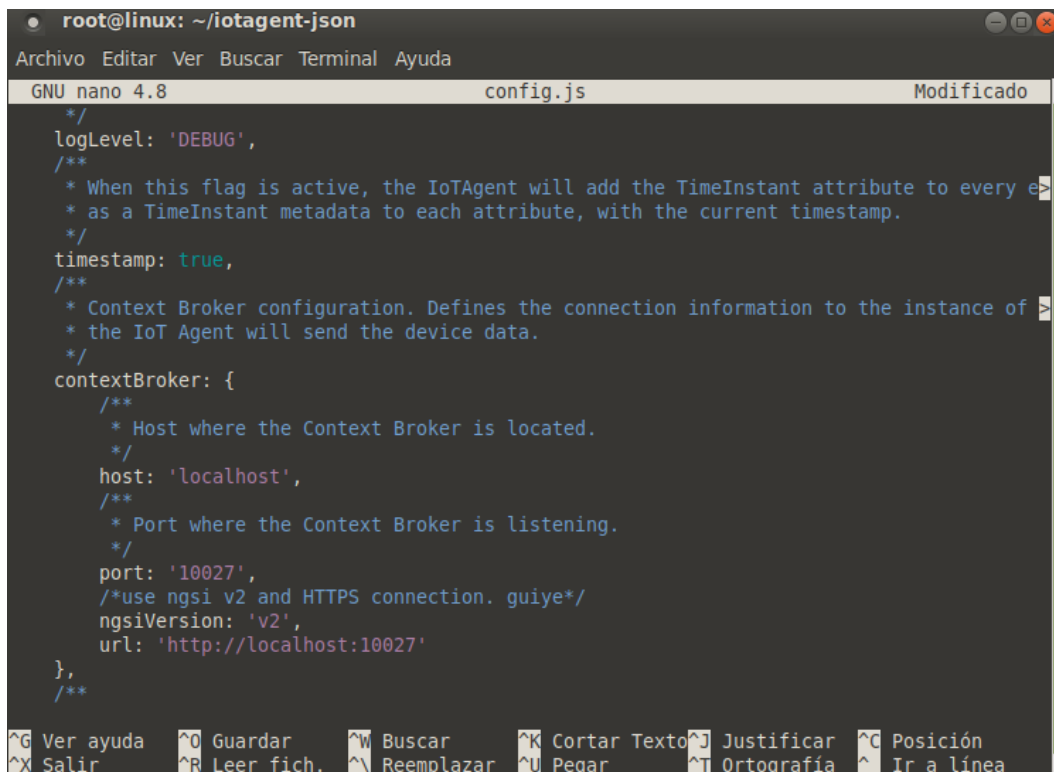
Lo primero que haremos será arrancar MongoDB, requisito necesario para hacer funcionar toda la plataforma. Después iniciaremos Orion, especificándole que vamos a usar el puerto 10027, para así evitar el colapso del puerto por defecto 1026.

```

>> service mongod start
>> systemctl start contextBroker
>> contextBroker -port 10027

```

A continuación, especificamos a IOTA que OCB va a usar el puerto 10027 y mediante HTTP.



```
root@linux: ~/iotagent-json
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 4.8 config.js Modificado
*/
logLevel: 'DEBUG',
/**
 * When this flag is active, the IoTAgent will add the TimeInstant attribute to every e
 * as a TimeInstant metadata to each attribute, with the current timestamp.
 */
timestamp: true,
/**
 * Context Broker configuration. Defines the connection information to the instance of
 * the IoT Agent will send the device data.
 */
contextBroker: {
  /**
   * Host where the Context Broker is located.
   */
  host: 'localhost',
  /**
   * Port where the Context Broker is listening.
   */
  port: '10027',
  /*use ngsi v2 and HTTPS connection. guiye*/
  ngsiVersion: 'v2',
  url: 'http://localhost:10027'
},
/**
^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Texto ^J Justificar ^C Posición
^X Salir ^R Leer fich. ^E Reemplazar ^U Pegar ^T Ortografía ^A Ir a línea
```

Ilustración 23 - Fichero de configuración de IOTA para el caso HTTP

Una vez comprobada que existe la conexión, nos encargaremos de llevar a cabo la consulta de las entidades existentes en el Orion en este momento. Para ello voy a realizar una petición de Consulta a Orion:



```
root@linux:~# curl -iX GET http://localhost:10027/v2/entities/
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Length: 941
Content-Type: application/json
Fiware-Correlator: 44c2a998-09ab-11ec-a13e-c918731f0f2a
Date: Mon, 30 Aug 2021 15:59:33 GMT

[{"id":"Room2","type":"Room","pressure":{"type":"Number","value":720,"metadata":{}}, "temperature":{"type":"Number","value":23,"metadata":{}}, {"id":"urn:ngsi-ld:Motion:001","type":"Motion","TimeInstant":{"type":"IS08601","value":"2021-07-19T10:36:00.771Z","metadata":{}}, "count":{"type":"Integer","value":1,"metadata":{"TimeInstant":{"type":"IS08601","value":"2021-07-19T10:36:00.771Z"}}}, "refStore":{"type":"Relationship","value":"urn:ngsi-ld:Store:001","metadata":{"TimeInstant":{"type":"IS08601","value":"2021-07-19T10:36:00.771Z"}}}, {"id":"urn:ngsi-ld:Motion:002","type":"Motion","TimeInstant":{"type":"DateTime","value":"2021-08-30T15:42:40.104Z","metadata":{}}, "count":{"type":"Integer","value":3,"metadata":{"TimeInstant":{"type":"DateTime","value":"2021-08-30T15:42:40.104Z"}}}, "refStore":{"type":"Relationship","value":"urn:ngsi-ld:Store:002","metadata":{"TimeInstant":{"type":"DateTime","value":"2021-08-30T15:42:40.104Z"}}}]
```

Ilustración 24-Petición de Consulta de Entidades a Orion

Al recibir la respuesta, vemos en texto plano las entidades que actualmente se encuentran alojadas, que son tres: *Room2*, *urn:ngsi-ld:Motion:001* y *urn:ngsi-ld:Motion:002*.

Ahora es el turno de analizar esta petición, mediante Wireshark. Captamos todos los paquetes y buscamos el que sale del puerto 10027 con los datos. Comprobamos que se encuentra también en texto plano en el transcurso de la red, y que podemos perfectamente diferenciar en ellos las entidades.

6	0.000451159	127.0.0.1	127.0.0.1	SSL	119	Continuation Data
7	0.000587908	127.0.0.1	127.0.0.1	SSL	106	Continuation Data
8	0.000591129	127.0.0.1	127.0.0.1	TCP	68	51226 → 27017 [ACK] Seq=52 Ack=39 Win=512 Len=0 TSval=2163522...
9	0.000653104	127.0.0.1	127.0.0.1	TCP	222	51226 → 27017 [PSH, ACK] Seq=52 Ack=39 Win=512 Len=154 TSval=...
10	0.000904616	127.0.0.1	127.0.0.1	TCP	2088	27017 → 51226 [PSH, ACK] Seq=39 Ack=206 Win=512 Len=2020 TSva...
11	0.000908804	127.0.0.1	127.0.0.1	TCP	68	51226 → 27017 [ACK] Seq=206 Ack=2059 Win=500 Len=0 TSval=2163...
12	0.001073730	127.0.0.1	127.0.0.1	TCP	653	51226 → 27017 [PSH, ACK] Seq=206 Ack=2059 Win=512 Len=585 TSv...
13	0.001211892	127.0.0.1	127.0.0.1	TCP	176	27017 → 51226 [PSH, ACK] Seq=2059 Ack=791 Win=512 Len=108 TSv...
14	0.001215529	127.0.0.1	127.0.0.1	TCP	68	51226 → 27017 [ACK] Seq=791 Ack=2167 Win=512 Len=0 TSval=2163...
15	0.001356269	127.0.0.1	127.0.0.1	TCP	258	10027 → 46286 [PSH, ACK] Seq=1 Ack=92 Win=65536 Len=190 TSval...
16	0.001362393	127.0.0.1	127.0.0.1	TCP	68	46286 → 10027 [ACK] Seq=92 Ack=191 Win=65408 Len=0 TSval=2163...
17	0.001485649	127.0.0.1	127.0.0.1	HTTP	1009	HTTP/1.1 200 OK (application/json)
18	0.001488071	127.0.0.1	127.0.0.1	TCP	68	46286 → 10027 [ACK] Seq=92 Ack=1132 Win=64512 Len=0 TSval=216...
19	0.001543860	127.0.0.1	127.0.0.1	TCP	68	46286 → 10027 [FIN, ACK] Seq=92 Ack=1132 Win=65536 Len=0 TSva...
20	0.002160030	127.0.0.1	127.0.0.1	TCP	68	10027 → 46286 [FIN, ACK] Seq=1132 Ack=93 Win=65536 Len=0 TSva...
21	0.002164043	127.0.0.1	127.0.0.1	TCP	68	46286 → 10027 [ACK] Seq=93 Ack=1133 Win=65536 Len=0 TSval=216...
22	0.582917774	127.0.0.1	127.0.0.1	TCP	76	51864 → 5037 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=...

▶ Frame 17: 1009 bytes on wire (8072 bits), 1009 bytes captured (8072 bits) on interface any, id 0  
 ▶ Linux cooked capture  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▼ Transmission Control Protocol, Src Port: 10027, Dst Port: 46286, Seq: 191, Ack: 92, Len: 941  
 Source Port: 10027  
 Destination Port: 46286

```

0000 00 00 03 04 00 06 00 00 00 00 00 00 00 00 08 00 .....
0010 45 00 03 e1 6f 28 40 00 40 06 c9 ec 7f 00 00 01 E...o(@. @.....
0020 7f 00 00 01 27 2b b4 ce 37 a7 98 e1 77 48 df 56 ....'+... 7...wH-V
0030 80 18 02 00 01 d6 00 00 01 01 08 0a 80 f4 ba d1 .....
0040 80 f4 ba d1 5b 7b 22 69 64 22 3a 22 52 6f 6f 6d ....[{"i d":"Room
0050 32 22 2c 22 74 79 70 65 22 3a 22 52 6f 6f 6d 22 2", "type ":"Room"
0060 2c 22 70 72 65 73 73 75 72 65 22 3a 7b 22 74 79 , "pressu re":{"ty
0070 70 65 22 3a 22 4e 75 6d 62 65 72 22 2c 22 76 61 pe":{"Num ber","va
0080 6c 75 65 22 3a 37 32 30 2c 22 6d 65 74 61 64 61 lue":720, "metada
0090 74 61 22 3a 7b 7d 7d 2c 22 74 65 6d 70 65 72 61 ta":{}}, "tempera
00a0 74 75 72 65 22 3a 7b 22 74 79 70 65 22 3a 22 4e ture":{" type":"N
00b0 75 6d 62 65 72 22 2c 22 76 61 6c 75 65 22 3a 32 umber", " value":2
00c0 33 2c 22 6d 65 74 61 64 61 74 61 22 3a 7b 7d 7d 3, "metad ata":{}}
00d0 7d 2c 7b 22 69 64 22 3a 22 75 72 6e 3a 6e 67 73 }, {"id": "urn:ngs
00e0 69 2d 6c 6a 3a 4d 6f 74 69 6f 6e 3a 30 30 31 22 1-d:Mot ion:001"
00f0 2c 22 74 79 70 65 22 3a 22 4d 6f 74 69 6f 6e 22 , "type": "Motion"
0100 2c 22 54 69 6d 65 49 6e 73 74 61 6e 74 22 3a 7b , "TimeIn stant":{
0110 22 74 79 70 65 22 3a 22 49 53 4f 38 36 30 31 22 "type": " IS08601"
0120 2c 22 76 61 6c 75 65 22 3a 22 32 30 32 31 2d 30 , "value" : "2021-0
0130 37 2d 31 39 54 31 30 3a 33 36 3a 30 30 2e 37 37 7-19T10: 36:00.77
0140 31 5a 22 2c 22 6d 65 74 61 64 61 74 61 22 3a 7b 1Z", "met adata":{
0150 7d 7d 2c 22 63 6f 75 6e 74 22 3a 7b 22 74 79 70 }, {"coun t":{"typ
0160 65 22 3a 22 49 6e 74 65 67 65 72 22 2c 22 76 61 e": "Inte ger", "va
0170 6c 75 65 22 3a 22 31 22 2c 22 6d 65 74 61 64 61 lue": "1", "metada
0180 74 61 22 3a 7b 22 54 69 6d 65 49 6e 73 74 61 6e ta":{"Ti meInstan
0190 74 22 3a 7b 22 74 79 70 65 22 3a 22 49 53 4f 38 t":{"typ e": "IS08
01a0 36 30 31 22 2c 22 76 61 6c 75 65 22 3a 22 32 30 601", "va lue": "20
01b0 32 31 2d 30 37 2d 31 39 54 31 30 3a 33 36 3a 30 21-07-19 T10:36:0
01c0 30 2e 37 37 31 5a 22 7d 7d 7d 2c 22 72 65 66 53 0.771Z"} }, "refs
01d0 74 6f 72 65 22 3a 7b 22 74 79 70 65 22 3a 22 52 tore":{" type": "R
01e0 65 6c 61 74 69 6f 6e 73 68 69 70 22 2c 22 76 61 elations hip", "va
01f0 6c 75 65 22 3a 22 75 72 6e 3a 6e 67 73 69 2d 6c lue": "ur n:ngsi-l
0200 64 3a 53 74 6f 72 65 3a 30 30 31 22 2c 22 6d 65 d:Store: 001", "me
0210 74 61 64 61 74 61 22 3a 7b 22 54 69 6d 65 49 6e tadata": {"TimeIn
0220 73 74 61 6e 74 22 3a 7b 22 74 79 70 65 22 3a 22 stant":{" type": "
0230 49 53 4f 38 36 30 31 22 2c 22 76 61 6c 75 65 22 IS08601", "value"
0240 3a 22 32 30 32 31 2d 30 37 2d 31 39 54 31 30 3a : "2021-0 7-19T10:
  
```

Ilustración 25 - Petición de consulta captada por Wireshark

Vamos ahora a analizar la modificación de datos mediante IOTA, para este caso enviaremos una petición HTTP a IOTA simulando que somos el sensor. En este caso, vamos a utilizar la entidad *motion002* para modificar su dato *count*.

```

root@linux:~# curl -iX POST 'http://localhost:7896/iot/json?k=jggokgpepnvbs2uv4s40d59ov2&i=motion002' -H '
Content-Type: application/json' -d '{"c": "2"}'
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 0
ETag: W/"0-2j mj7l5rSw0yVb/vlWAYkK/YBwk"
Date: Mon, 30 Aug 2021 16:03:20 GMT
Connection: keep-alive
  
```

Ilustración 26- Simulación de Modificación del contexto por parte del sensor

Esta petición irá en HTTP y no entra en nuestro escenario securizarla. Sin embargo, nos encargaremos de captar mediante Wireshark el paquete que envía nuestro nuevo valor IOTA a Orion.

```

53 1.632931828 127.0.0.1 127.0.0.1 TCP 69 52106 -- 27017 [PSH, ACK] Seq=609 Ack=1091 Win=512 Len=1 TSval=
54 1.632940792 127.0.0.1 127.0.0.1 TCP 68 27017 -- 52106 [ACK] Seq=1091 Ack=610 Win=512 Len=0 TSval=2163...
55 1.632958787 127.0.0.1 127.0.0.1 TCP 271 52106 -- 27017 [PSH, ACK] Seq=610 Ack=1091 Win=512 Len=203 TSv...
56 1.633090122 127.0.0.1 127.0.0.1 TCP 176 27017 -- 52106 [PSH, ACK] Seq=1091 Ack=813 Win=512 Len=108 TSv...
57 1.633094017 127.0.0.1 127.0.0.1 TCP 68 52106 -- 27017 [ACK] Seq=813 Ack=1199 Win=512 Len=0 TSval=2163...
58 1.638908433 127.0.0.1 127.0.0.1 TCP 76 47632 -- 10027 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM...
59 1.638918046 127.0.0.1 127.0.0.1 TCP 76 10027 -- 47632 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=6549...
60 1.638925163 127.0.0.1 127.0.0.1 TCP 68 47632 -- 10027 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2163748...
61 1.639040841 127.0.0.1 127.0.0.1 HTTP 649 PATCH /v2/entities/urn:ngsi-ld:Motion:002/attns?type=Motion H...
62 1.639044573 127.0.0.1 127.0.0.1 TCP 68 10027 -- 47632 [ACK] Seq=1 Ack=582 Win=65024 Len=0 TSval=21637...
63 1.639296474 127.0.0.1 127.0.0.1 SSL 119 Continuation Data
64 1.639422063 127.0.0.1 127.0.0.1 SSL 106 Continuation Data
65 1.639425419 127.0.0.1 127.0.0.1 TCP 68 51226 -- 27017 [ACK] Seq=52 Ack=39 Win=512 Len=0 TSval=2163748...
66 1.639469693 127.0.0.1 127.0.0.1 SSL 345 Continuation Data
67 1.639775717 127.0.0.1 127.0.0.1 SSL 195 Continuation Data
68 1.639779691 127.0.0.1 127.0.0.1 TCP 68 51226 -- 27017 [ACK] Seq=329 Ack=166 Win=512 Len=0 TSval=21637...
69 1.639841244 127.0.0.1 127.0.0.1 TCP 256 51226 -- 27017 [PSH, ACK] Seq=329 Ack=166 Win=512 Len=188 TSva...

```

▶ Frame 61: 649 bytes on wire (5192 bits), 649 bytes captured (5192 bits) on interface any, id 0  
▶ Linux cooked capture  
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
▼ Transmission Control Protocol, Src Port: 47632, Dst Port: 10027, Seq: 1, Ack: 1, Len: 581  
Source Port: 47632  
Destination Port: 10027

```

0030 80 18 02 00 00 06 00 00 01 01 08 0a 80 f8 2e 3f .....n.....?
0040 80 f8 2e 3f 50 41 54 43 48 20 2f 76 32 2f 65 6e ...?PATC H /v2/en
0050 74 69 74 69 65 73 2f 75 72 6e 3a 6e 67 73 69 2d tities/u rn:ngsi-
0060 6c 64 3a 4d 6f 74 69 6f 6e 3a 30 30 32 2f 61 74 ld:Motio n:002/at
0070 74 72 73 3f 74 79 70 65 3d 4d 6f 74 69 6f 6e 20 trs?type =Motion
0080 48 54 54 50 2f 31 2e 31 0d 0a 66 69 77 61 72 65 HTTP/1.1 f:fiware
0090 2d 73 65 72 76 69 63 65 3a 20 6f 70 65 6e 69 6f -service : openio
00a0 74 0d 0a 66 69 77 61 72 65 2d 73 65 72 76 69 63 t:fiwar e-servic
00b0 65 70 61 74 68 3a 20 2f 0d 0a 68 6f 73 74 3a 20 epath: / :-host:
00c0 6c 6f 63 61 6c 68 6f 73 74 3a 31 30 30 32 37 0d localhos t:10027
00d0 0a 61 63 63 65 70 74 3a 20 61 70 70 6c 69 63 61 -accept: applica
00e0 74 69 6f 6e 2f 6a 73 6f 6e 0d 0a 63 6f 6e 74 65 tion/jso n:conte
00f0 6e 74 2d 74 79 70 65 3a 20 61 70 70 6c 69 63 61 nt-type: applica
0100 74 69 6f 6e 2f 6a 73 6f 6e 0d 0a 63 6f 6e 74 65 tion/jso n:conte
0110 6e 74 2d 6c 65 6e 67 74 68 3a 20 33 34 30 0d 0a nt-lengt h: 340
0120 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 63 6c 6f 73 Connection: clos
0130 65 0d 0a 0d 0a 7b 22 63 6f 75 6e 74 22 3a 7b 22 e...{"count":{"t
0140 74 79 70 65 22 3a 22 49 6e 74 65 67 65 72 22 2c type":"Integer",
0150 22 76 61 6c 75 65 22 3a 22 32 22 2c 22 6d 65 74 "value": "2", "met
0160 61 64 61 74 61 22 3a 7b 22 54 69 6d 65 49 6e 73 adata":{"TimeIns
0170 74 61 6e 74 22 3a 7b 22 74 79 70 65 22 3a 22 44 tant":{" type":"D
0180 61 74 65 54 69 6d 65 22 2c 22 76 61 6c 75 65 22 ateTime", "value"
0190 3a 22 32 30 32 31 2d 30 38 2d 33 30 54 31 36 3a ":2021-08-30T16:
01a0 30 33 3a 32 30 2e 30 38 38 5a 22 7d 7d 2c 22 03:20.08 8Z"}}, "
01b0 72 65 66 53 74 6f 72 65 22 3a 7b 22 74 79 70 65 refStore ":{"type
01c0 22 3a 22 52 65 6c 61 74 69 6f 6e 73 68 69 70 22 ":":"Relat ionship"
01d0 2c 22 76 61 6c 75 65 22 3a 22 75 72 6e 3a 6e 67 , "value" : "urn:ng
01e0 73 69 2d 6c 64 3a 53 74 6f 72 65 3a 30 30 32 22 si-ld:St ore:002"
01f0 2c 22 6d 65 74 61 64 61 74 61 22 3a 7b 22 54 69 , "metada ta":{"Ti
0200 6d 65 49 6e 73 74 61 6e 74 22 3a 7b 22 74 79 70 meInstan t":{"typ
0210 65 22 3a 22 44 61 74 65 54 69 6d 65 22 2c 22 76 e":"Date Time", "v
0220 61 6c 75 65 22 3a 22 32 30 32 31 2d 30 38 2d 33 alue":"2 021-08-3
0230 30 54 31 36 3a 30 33 3a 32 30 2e 30 38 38 5a 22 0T16:03: 20.088Z"
0240 7d 7d 7d 2c 22 54 69 6d 65 49 6e 73 74 61 6e 74 }}, "Tim eInstant
0250 22 3a 7b 22 74 79 70 65 22 3a 22 44 61 74 65 54 ": {"type "":"DateT
0260 69 6d 65 22 2c 22 76 61 6c 75 65 22 3a 22 32 30 ime", "va lue":"20
0270 32 31 2d 30 38 2d 33 30 54 31 36 3a 30 33 3a 32 21-08-30 T16:03:2
0280 30 2e 30 38 38 5a 22 7d 7d 0.088Z"} }

```

Ilustración 27-Captura del paquete que envía el valor que actualiza el sensor

En este caso comprobamos que el valor se encuentra totalmente expuesto a una captura por parte de un agente ajeno. Queda comprobado que es necesaria una encriptación, pero lo más importante es que ahora vamos a demostrar que lo hemos conseguido.

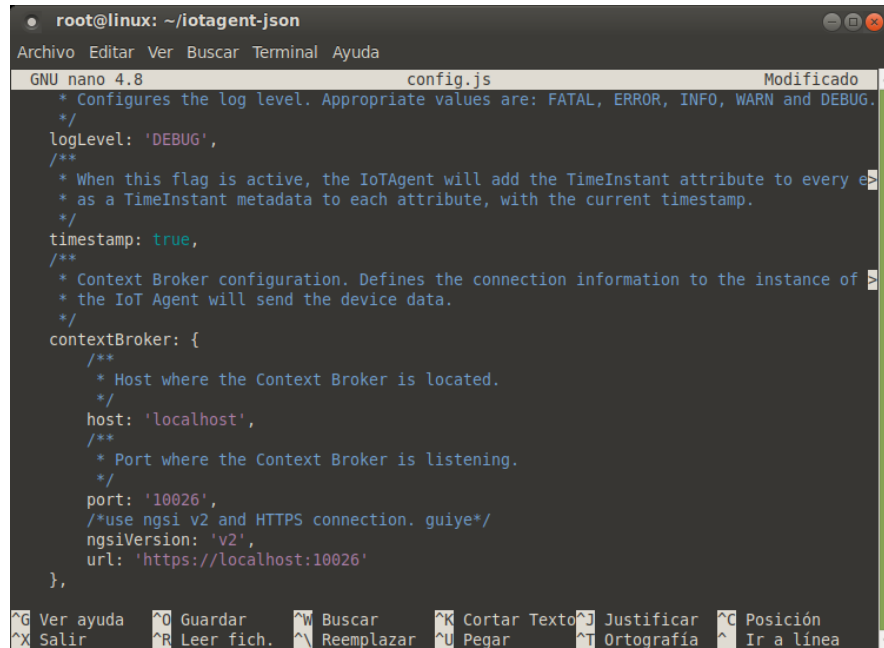
Para ello vamos a parar los procesos de OCB e IOTA, y vamos a volver a arrancarlos de forma securizada. Empezamos ejecutando Orion con sus comandos pertinentes, en un nuevo puerto.

```

>> systemctl start contextBroker
>> contextBroker -port 10026 -https -key localhost.key -cert localhost.pem

```

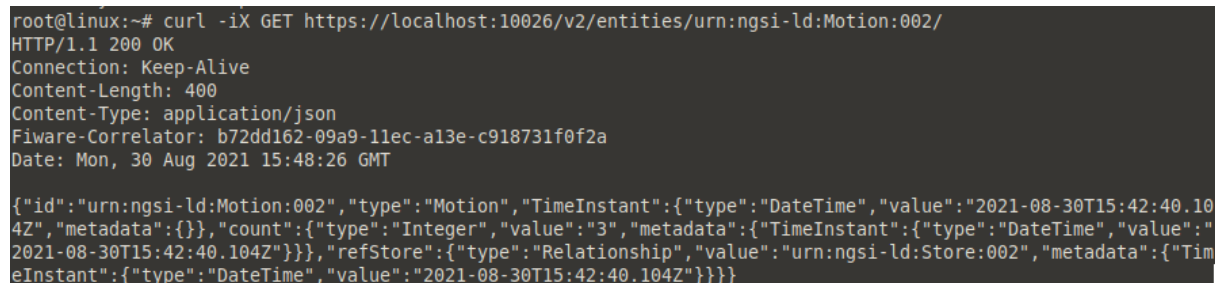
Después, modificaremos el fichero de configuración de IOTA para establecer la nueva dirección HTTPS a la que acudir para contactar con OCB.



```
root@linux: ~/iotagent-json
Archivo Editar Ver Buscar Terminal Ayuda
GNU nano 4.8 config.js Modificado
* Configures the log level. Appropriate values are: FATAL, ERROR, INFO, WARN and DEBUG.
*/
logLevel: 'DEBUG',
/**
 * When this flag is active, the IoTAgent will add the TimeInstant attribute to every e>
 * as a TimeInstant metadata to each attribute, with the current timestamp.
 */
timestamp: true,
/**
 * Context Broker configuration. Defines the connection information to the instance of >
 * the IoT Agent will send the device data.
 */
contextBroker: {
  /**
   * Host where the Context Broker is located.
   */
  host: 'localhost',
  /**
   * Port where the Context Broker is listening.
   */
  port: '10026',
  /*use ngsi v2 and HTTPS connection. guiye*/
  ngsiVersion: 'v2',
  url: 'https://localhost:10026'
},
^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Texto ^J Justificar ^C Posición
^X Salir ^R Leer fich. ^M Reemplazar ^U Pegar ^T Ortografía ^_ Ir a línea
```

Ilustración 28 - Modificación del fichero de configuración de IOTA para HTTPS

Una vez hecho el trabajo previo, se va a realizar una consulta a OCB sobre las entidades que contiene, concretamente sobre *urn:ngsi-ld:Motion:002*. Así, recibiremos como respuesta la entidad en el mismo formato que cuando lo hicimos sin securizar la plataforma.



```
root@linux:~# curl -iX GET https://localhost:10026/v2/entities/urn:ngsi-ld:Motion:002/
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Length: 400
Content-Type: application/json
Fiware-Correlator: b72dd162-09a9-11ec-a13e-c918731f0f2a
Date: Mon, 30 Aug 2021 15:48:26 GMT

{"id":"urn:ngsi-ld:Motion:002","type":"Motion","TimeInstant":{"type":"DateTime","value":"2021-08-30T15:42:40.104Z","metadata":{}}, "count":{"type":"Integer","value":"3","metadata":{"TimeInstant":{"type":"DateTime","value":"2021-08-30T15:42:40.104Z"}}}, "refStore":{"type":"Relationship","value":"urn:ngsi-ld:Store:002","metadata":{"TimeInstant":{"type":"DateTime","value":"2021-08-30T15:42:40.104Z"}}}}
```

Ilustración 29 - Petición de Consulta a OCB mediante HTTPS

Esta petición la vamos a capturar en Wireshark, y podemos comprobar que, tras el establecimiento de TLS, todos los mensajes intercambiados debido a la petición se empiezan a realizar de forma encriptada.

```

698 39.808807180 127.0.0.1 127.0.0.1 TCP 68 50920 -> 10026 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2162855..
699 39.816384296 127.0.0.1 127.0.0.1 TLSv1.3 585 Client Hello
700 39.816394861 127.0.0.1 127.0.0.1 TCP 68 10026 -> 50920 [ACK] Seq=1 Ack=518 Win=65024 Len=0 TSval=21628..
701 39.816583216 127.0.0.1 127.0.0.1 TLSv1.3 201 Server Hello, Change Cipher Spec
702 39.818279518 127.0.0.1 127.0.0.1 TLSv1.3 1436 Application Data, Application Data, Application Data, Applica..
703 39.818328822 127.0.0.1 127.0.0.1 TCP 68 50920 -> 10026 [ACK] Seq=518 Ack=134 Win=65408 Len=0 TSval=216..
704 39.818331643 127.0.0.1 127.0.0.1 TCP 68 50920 -> 10026 [ACK] Seq=518 Ack=1502 Win=64128 Len=0 TSval=21..
705 39.818778881 127.0.0.1 127.0.0.1 TLSv1.3 148 Change Cipher Spec, Application Data
706 39.818815557 127.0.0.1 127.0.0.1 TCP 68 10026 -> 50920 [ACK] Seq=1502 Ack=598 Win=65536 Len=0 TSval=21..
707 39.818866004 127.0.0.1 127.0.0.1 TLSv1.3 204 Application Data
708 39.818867987 127.0.0.1 127.0.0.1 TCP 68 10026 -> 50920 [ACK] Seq=1502 Ack=734 Win=65408 Len=0 TSval=21..
709 39.819133139 127.0.0.1 127.0.0.1 SSL 119 Continuation Data
710 39.819262130 127.0.0.1 127.0.0.1 SSL 106 Continuation Data

```

▶ Frame 707: 204 bytes on wire (1632 bits), 204 bytes captured (1632 bits) on interface any, id 0  
▶ Linux cooked capture  
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
▼ Transmission Control Protocol, Src Port: 50920, Dst Port: 10026, Seq: 598, Ack: 1502, Len: 136  
Source Port: 50920  
Destination Port: 10026

```

0000 00 00 03 04 00 06 00 00 00 00 00 00 68 00 08 00 .....h...
0010 45 00 00 bc 66 ba 40 00 40 06 d5 7f 7f 00 00 01 E...f.@...
0020 7f 00 00 01 c6 e8 27 2a ee 3b f2 3d eb 52 0b a6 .....*...;=R...
0030 80 18 02 00 fe b0 00 00 01 01 08 0a 80 ea 8d 38 .....8...
0040 80 ea 8d 38 17 03 03 00 83 c7 c7 64 20 86 77 63 ...8...d..wc
0050 0d 59 e4 cc 55 1a 3f 1a 67 fb 90 f9 de 24 90 39 .Y..U.?..g...$ 9
0060 38 c6 32 97 c8 85 b1 f1 ac 5d 68 0c 7c 81 bc 60 8.2.....]h|...
0070 8e 70 0e 87 48 cc d5 ba 1a a9 af 81 c1 dd ce e6 .p..H.....
0080 f4 1e ec f5 4c 5f bd 3f 56 80 b6 f4 6a a1 33 01 ...L..? V...j:3
0090 4c 55 e4 0a 67 4f 95 01 f4 4b 7b 8c 6b 2a 3a 86 LU..g0...K{.k*:
00a0 b2 21 24 21 2d b4 5c 8f 5b b7 fa d1 b5 69 c0 33 .!$!-\.. \ [...i:3
00b0 f5 5b 1f 91 59 f7 53 14 fb 69 0a f4 b4 b5 2e ba .[...Y.S..i.....
00c0 5b 61 b8 6b 9e ea 25 68 7f 0d ae 3e [a.k.%h...>

```

Ilustración 30 - Captura mediante Wireshark de un paquete de intercambio de información en la Consulta a OCB

Ahora es el turno de comprobar que la conexión IOTA-OCB se establece de manera encriptada. Para ello vamos a simular de nuevo, y con el mismo comando, al sensor. Puesto que esta conexión no es la que debe cambiar.

```

root@linux:~# curl -iX POST 'http://localhost:7896/iot/json?k=4jggokgpepnvsb2uv4s40d59ov2&i=motion002' -H '
Content-Type: application/json' -d '{"c": "3"}'
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 0
ETag: W/"0-2jmj7l5rSw0yVb/vlWAYkK/YBwk"
Date: Mon, 30 Aug 2021 15:42:40 GMT
Connection: keep-alive

```

Ilustración 31 - Simulación de Modificación del contexto por parte del sensor

Esa petición la capturamos mediante Wireshark y podemos comprobar que la conexión entre el sensor e IOTA no está encriptada, y, sin embargo, la comunicación entre IOTA y OCB sí se encuentra encriptada.



80	5.059715255	127.0.0.1	127.0.0.1	TCP	68 43392 → 7896 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=21625084...
81	5.059770617	127.0.0.1	127.0.0.1	HTTP	259 POST /iot/json?k=4jggokgpepnvsb2uv4s40d59ov2&i=motion002 HTTP/1.1
82	5.059775350	127.0.0.1	127.0.0.1	TCP	68 7896 → 43392 [ACK] Seq=1 Ack=192 Win=65408 Len=0 TSval=216250...
83	5.063284329	127.0.0.1	127.0.0.1	SSL	88 Continuation Data
84	5.063358378	127.0.0.1	127.0.0.1	TCP	69 35206 → 27017 [PSH, ACK] Seq=73 Ack=305 Win=512 Len=1 TSval=2...
85	5.063374381	127.0.0.1	127.0.0.1	TCP	68 27017 → 35206 [ACK] Seq=305 Ack=74 Win=512 Len=0 TSval=216250...
86	5.063390870	127.0.0.1	127.0.0.1	SSL	318 Continuation Data
87	5.063892661	127.0.0.1	127.0.0.1	TCP	320 27017 → 35206 [PSH, ACK] Seq=305 Ack=324 Win=512 Len=252 TSva...
88	5.063898402	127.0.0.1	127.0.0.1	TCP	68 35206 → 27017 [ACK] Seq=324 Ack=557 Win=511 Len=0 TSval=21625...
89	5.065608126	127.0.0.1	127.0.0.1	SSL	88 Continuation Data

▶ Frame 81: 259 bytes on wire (2072 bits), 259 bytes captured (2072 bits) on interface any, id 0  
 ▶ Linux cooked capture  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▼ Transmission Control Protocol, Src Port: 43392, Dst Port: 7896, Seq: 1, Ack: 1, Len: 191  
 Source Port: 43392

```

0000 00 00 03 04 00 06 00 00 00 00 00 03 07 08 00 .....
0010 45 00 00 f3 ba bc 40 00 40 06 81 46 7f 00 00 01 E.....@.F.....
0020 7f 00 00 01 a9 80 1e d8 9c 2a e9 92 93 da 82 80 .....*]u.....
0030 80 18 02 00 fe e7 00 00 01 01 08 0a 80 e5 42 86 .....B.....
0040 80 e5 42 85 50 4f 53 54 20 2f 69 6f 74 2f 6a 73 ..B POST /iot/js
0050 6f 6e 3f 6b 3d 3a 6a 67 67 6f 6b 67 70 65 70 6e on?k=4jg gokgpepn
0060 76 73 62 32 75 76 34 73 34 30 64 35 39 6f 76 32 vsb2uv4s 40d59ov2
0070 26 69 3d 6d 6f 74 69 6f 6e 30 30 32 20 48 54 54 &i=motio n002 HTT
0080 50 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 6c 6f 63 P/1.1. Host: loc
0090 61 6c 68 6f 73 74 3a 37 38 39 36 0d 0a 55 73 65 alhost:7 896 Use
00a0 72 2d 41 67 65 6e 74 3a 20 63 75 72 6c 2f 37 2e r-Agent: curl/7.
00b0 36 38 2e 30 0d 0a 41 63 63 65 70 74 3a 20 2a 2f 68.0. Ac cept: /*
00c0 2a 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a *.Conte nt-Type:
00d0 20 61 70 70 6c 69 63 61 74 69 6f 6e 2f 6a 73 6f applica tion/jso
00e0 6e 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 n-Conte nt-Lengt
00f0 68 3a 20 31 30 0d 0a 0d 0a 7b 22 63 22 3a 20 22 h: 10... {"c": "
0100 33 22 7d 3"}
  
```

Ilustración 32 - Petición que simula al sensor capturada mediante Wireshark

103	5.070549735	127.0.0.1	127.0.0.1	TCP	68 45242 → 10026 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2162508...
104	5.070729843	127.0.0.1	127.0.0.1	TLV1.2	362 Client Hello
105	5.070733402	127.0.0.1	127.0.0.1	TCP	68 10026 → 45242 [ACK] Seq=1 Ack=295 Win=65280 Len=0 TSval=21625...
106	5.072559267	127.0.0.1	127.0.0.1	TLV1.2	1438 Server Hello, Certificate, Server Key Exchange, Server Hello ...
107	5.072564553	127.0.0.1	127.0.0.1	TCP	68 45242 → 10026 [ACK] Seq=295 Ack=1371 Win=64256 Len=0 TSval=21...
108	5.072812805	127.0.0.1	127.0.0.1	TLV1.2	161 Client Key Exchange, Change Cipher Spec, Encrypted Handshake ...
109	5.072828278	127.0.0.1	127.0.0.1	TCP	68 10026 → 45242 [ACK] Seq=1371 Ack=388 Win=65536 Len=0 TSval=21...
110	5.074927310	127.0.0.1	127.0.0.1	TLV1.2	119 Change Cipher Spec, Encrypted Handshake Message
111	5.074932007	127.0.0.1	127.0.0.1	TCP	68 45242 → 10026 [ACK] Seq=388 Ack=1422 Win=65536 Len=0 TSval=21...
112	5.075351497	127.0.0.1	127.0.0.1	TLV1.2	678 Application Data
113	5.075411234	127.0.0.1	127.0.0.1	TCP	68 10026 → 45242 [ACK] Seq=1422 Ack=998 Win=65024 Len=0 TSval=21...
114	5.075678827	127.0.0.1	127.0.0.1	SSL	119 Continuation Data
115	5.075806580	127.0.0.1	127.0.0.1	SSL	106 Continuation Data

▶ Frame 112: 678 bytes on wire (5424 bits), 678 bytes captured (5424 bits) on interface any, id 0  
 ▶ Linux cooked capture  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▼ Transmission Control Protocol, Src Port: 45242, Dst Port: 10026, Seq: 388, Ack: 1422, Len: 610

```

0000 00 00 03 04 00 06 00 00 00 00 00 00 00 08 00 .....
0010 45 00 02 96 90 81 40 00 40 06 a9 de 7f 00 00 01 E.....@.F.....
0020 7f 00 00 01 b0 ba 27 2a 5d bf 75 9b e8 c3 b1 e5 .....*]u.....
0030 80 18 02 00 00 8b 00 00 01 01 08 0a 80 e5 42 95 .....B.....
0040 80 e5 42 95 17 03 03 02 5d cb 86 1b 61 8a a1 31 ..B.....] a-1
0050 28 e1 b8 a4 0f 7f 06 ae a6 45 dc 56 29 86 a5 ae (.....E V).....
0060 ae 10 b1 17 db 6d 86 b1 3a ca f0 54 ae 0a 3f bf .....m...:T?
0070 37 84 50 66 c7 00 b2 5e 9b bf 15 27 80 91 be e9 7.Pf...^.....
0080 11 14 21 7f 9a 68 1a 53 af 4c 40 b1 b8 0d 6c 93 ...!h s L@-1.
0090 1b f2 b9 74 80 fb 42 56 ac d7 60 6e 75 66 6b 8d ...t BV...nufk
00a0 df ca 0c 9a b5 5f a0 85 5c ba a2 0a 4a 09 24 72 ..... \...J.Sr
00b0 a8 0f 0b fe 8d bb 12 59 ae 8d fa dc e3 69 22 9d .....Y...i"
00c0 ee 9d dc dc 02 21 ed 2a 81 3a 87 e5 79 8e ac 1d ...!...:y...
00d0 33 ba ee 4d 75 88 1e ef 57 86 58 e9 2a 02 78 b5 3.Mu...W.X*x
00e0 67 16 56 ee 5d 9a c2 c9 ce 99 f2 31 50 39 f7 1c g.V]...-1P9
00f0 16 4e 97 4c 7f bb 2b 2a f8 48 df aa ed a5 60 45 N.L+*H...E
0100 3d 2a ce 55 9d 9b 17 8c 9a 34 97 99 78 7f 3a ea =*U...4-x:
0110 0d 5f 58 98 c1 43 d2 64 f1 ad 9c 81 2c ef 37 54 _[...C d...7T
0120 9a d1 b4 d4 e8 07 42 32 c0 de 6c 46 3e ee 0f f4 ...B2...1F>
0130 24 2f bd 95 6e 01 41 ef 1f f9 9a 5b 17 7a 64 99 S/n-A...[zd
0140 0b 06 95 8f 46 07 d0 d4 ac 22 54 bc 71 06 22 e4 ...F...Tq"
0150 ca e9 c1 45 39 9d 4f 91 d6 de cd 21 82 2b 88 b3 ...E9-O...!+
0160 22 be a9 a7 b9 f5 a3 a9 59 16 2c 41 9e 5f 97 eb "...Y...A...
0170 62 08 ea fb 4a 03 8c ce 1a 8f 07 cc 8b 7a 1e 6f b...J...Z o
0180 41 7d 09 17 0f 58 c0 6b 97 a7 e5 11 7a b7 3d 2c A}...X k...z =,
0190 f0 50 4e 8d c9 f0 8e 50 dd ed ff a7 8a 46 23 df PN...P...F#
01a0 c1 ea b9 79 92 b8 51 1d 8e 9b 27 21 7d 96 8f 6f ...y-Q...!} o
01b0 01 e4 93 ee 03 b9 08 69 a7 7b 01 32 12 78 3f 8e .....i...{2 x?
01c0 dc 21 4b 7a a5 68 98 17 a1 52 9b fc 49 29 c5 0c !Kz:h...R-I)
01d0 a4 f5 3c f3 13 f1 c7 7d f5 91 73 ab ff 58 88 b3 <...} s-X
01e0 47 21 70 41 cb 80 f6 d8 30 d7 62 6e 3a 68 d9 0a G!pA...0 bn:h
01f0 ec ad a8 59 10 f2 ba 49 a6 ab 39 71 9a 59 d1 9a ...Y...I -9q-Y
0200 7d 53 26 de d4 69 d4 09 67 c3 07 ac 93 89 c9 e2 }S&...i g...>Z
0210 5c 0d 13 15 2e e6 87 7d f9 d4 a1 88 bc 3e 5a ac ...}...>
0220 f1 df 84 45 ad 6f 7c 0a d8 2c ef 4a 8f 07 13 42 ...E o]...J-B
0230 56 b2 94 9f f3 4e ec ad 5c 22 a6 93 9d 1d 6d 27 V...N... \...m'
0240 8e d3 d3 12 7d f1 b5 c1 a3 e6 ef 17 ec ac ec b9 ...}...
0250 00 fd 8b 38 28 84 df 79 e9 b6 cf b6 1e 02 49 a3 ...8(y...-I
0260 38 63 43 7a d9 57 5c c9 ed 1e 9f b1 e4 4c a7 b1 8Cz W...L
0270 7f 28 53 2c d8 84 1f 33 c0 43 6e 17 c3 a5 c6 e1 (S...3 -Cn
0280 8a f6 a5 78 59 e3 49 21 e4 91 53 75 96 b0 18 16 ...xY-I! -Su
0290 b0 68 ac e4 fb 95 51 cd 00 a3 b7 8c c9 93 39 19 h...Q...-9
02a0 2b 8d 8d 8d 9f ff +.....
  
```

Ilustración 33 - Comunicación entre IOTA y OCB, capturada mediante Wireshark

Por tanto, hemos comprobado que se han conseguido encriptar las comunicaciones de la plataforma IOT FIWARE.

### 3.3.4 Comparativa de uso de recursos respecto a escenario sin encriptación

Una de las mayores problemáticas a la hora de llevar a cabo la encriptación en IOT es que los dispositivos suelen obligar a dar un uso ligero a su CPU, y se desea que estos requieran del menor gasto energético posible ya que tienen una potencia muy baja para así ampliar su autonomía, en casos inalámbricos, o su vida útil.

En este apartado se va a hacer una comparativa entre nuestros dos escenarios, con encriptación y sin ella, en la que se pretende explorar cuánto lastraría llevarla a cabo. Para ello, gracias a la herramienta Wireshark mediremos los tiempos en los que el sensor simulado envía el mensaje a IOTA para que actualice el contexto y éste responde de vuelta al sensor con el mensaje HTTP 200 OK. Después, restaremos estos tiempos mecánicamente para conocer cuánto se ha tardado en llevar la tarea. Esto lo realizaremos primero seis veces a la plataforma sin encriptar, y otras 6 a la encriptada mediante HTTPS.

Ilustración 34 - Captura de paquetes mediante Wireshark del proceso de actualización de contexto sin encriptar

En el caso de la plataforma sin encriptar hemos conseguido los siguientes tiempos:

Pruebas	Tiempos (s)
1	0.014311193
2	0.018038310
3	0.009376172
4	0.050200833
5	0.031868405
6	0.015852682

Tabla 3-2 – Tiempos conseguidos para la actualización de contexto sin encriptación

En este caso se van a descartar las pruebas 4 y 5, al resultarnos muy alejadas de la mediana. Obtendríamos una media de 14,39 milisegundos.

Ilustración 35 - Captura de paquetes mediante Wireshark del proceso de actualización de contexto encriptado mediante HTTPS

En el caso de la plataforma encriptada mediante HTTPS hemos conseguido los siguientes tiempos:

Pruebas	Tiempos
1	0.021037273
2	0.020972695
3	0.020181718
4	0.111035763
5	0.021292545
6	0.022951804

Tabla 3-3 – Tiempos conseguidos para la actualización de contexto con encriptación mediante HTTPS

En este caso se va a descartar la prueba 4 por encontrarse demasiado alejada de la mediana. Aquí obtendríamos una media de 21,27 milisegundos.

Podemos entonces comprobar que existe una latencia en el caso encriptado que exige que el dispositivo emplee aproximadamente un 47,86% más del tiempo procesando esta petición. No se puede estar seguro de que este tiempo sea completamente empleado por el procesador del dispositivo, pero gran parte sí. También se puede dar por cierto que, la mayoría del tiempo, el dispositivo se va a encontrar en reposo. Puesto que normalmente un sensor de este estilo no suele enviar datos cada segundo. Y puestos al caso, 7 milisegundos (diferencia que tienen cada caso) es menor que un 1% de 1 segundo y se puede considerar despreciable respecto a éste.

Por tanto, según esto creo que la encriptación es necesaria, implementable y útil; aunque quedaría pendiente llevar a cabo estos cálculos usando una plataforma IOT al uso, en la que poder usar sensores y dispositivos capacitados.

## 4 CONCLUSIONES Y LINEAS FUTURAS

---

A través de este trabajo, se ha intentado llevar a cabo un estudio sobre la securización de un enlace de comunicación en una plataforma IOT FIWARE. Tras haber implementado una encriptación HTTPS podemos decir que se trata de una implementación útil y necesaria para este tipo de comunicaciones, además de no ser excesivamente complicado su desarrollo. Añadir confidencialidad a nuestros datos en su paso por toda la red mundial nos permite tener una plataforma más robusta y menos accesible por parte de agentes ajenos a ella, que podrían beneficiarse de nuestra información.

Podemos decir que lo más enrevesado fue configurar la plataforma. Ya que la implementación de la misma se encuentra todavía en desarrollo y afinándose, más si cabe que resulta bastante dependiente del sistema operativo en el que se quiera instalar. Haber podido arrancarla mediante la aplicación de *kubernetes* (contenedores), *Docker*, sería más adecuado, pero su configuración también fue engorrosa y quitaba tiempo para llevar a cabo el desarrollo del trabajo.

Hubiera sido muy interesante haber podido implantar el escenario sanitario al completo, para poder darle más veracidad a nuestro planteamiento. Este consta de otros Generic Enablers como PEP Wilma, Authforce y Keyrock, que darían mucha más seguridad y consistencia a nuestra plataforma, convirtiéndola ya en una implementable en cualquier escenario profesional. Además, se ha realizado localmente y sería deseable haber podido implementarlo al menos entre dos máquinas diferentes. Sin embargo, no ha sido problema para poder desarrollarlo y hallar los resultados.

También por lo mismo, el haberle dado a Orion una IP fija nos hubiera ayudado a conseguir unos certificados no autofirmados que podrían haberle hecho accesible desde cualquier computadora. Sin necesidad de aportar la solución de que se confie ciegamente en la conexión. En términos de seguridad, este tema no se ve realmente correcto, o, por lo menos, optimizado.

Otra propuesta posiblemente implementable sería la de modificar Orion e IOTA para poder desarrollar en ellos la encriptación mediante AES. Perderíamos la eficiencia que nos aporta HTTPS, pero este modelo sería mucho más eficaz y deseable en cuanto a la confidencialidad de nuestros datos.

Por último, querría destacar las ventajas que nos aporta IOT y los resultados obtenidos en este trabajo para animar a su implementación con sensores y dispositivos físicos, y realizar en estos las mismas pruebas llevadas a cabo en este trabajo, para así continuar con los avances de ésta.

# 5 ANEXO A: INSTALACIÓN ORION CONTEXT BROKER

---

La instalación del Orion Context Broker puede resultar algo enrevesada, luego se va a dedicar este anexo a desarrollarla, para evitar dudas. En todo momento emplearemos el usuario `root`.

Lo primero será instalar MongoDB (<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/>):

```
>> wget -qO - https://www.mongodb.org/static/pgp/server-5.0.asc | sudo
apt-key add -

>> echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
focal/mongodb-org/5.0 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-5.0.list

>> sudo apt-get install -y mongodb-org

>>service mongod start
```

A continuación, dejo los comandos necesarios para llevar a cabo la instalación de Orion. Antes de la instalación, hay que llevar a cabo la instalación de las librerías necesarias:

Herramientas necesarias para la construcción:

```
>> sudo apt install build-essential cmake
```

Librerías generales, instalables mediante apt:

```
>> sudo apt install libboost-dev libboost-regex-dev libboost-thread-dev \
libboost-filesystem-dev libcurl4-gnutls-dev gnutls-dev \
libcrypt-dev libssl-dev uuid-dev libsasl2-dev
```

Mongo Driver:

```
>> wget https://github.com/mongodb/mongo-c-
driver/releases/download/1.17.4/mongo-c-driver-1.17.4.tar.gz

>>tar xfvz mongo-c-driver-1.17.4.tar.gz

>>cd mongo-c-driver-1.17.4

>>mkdir cmake-build

>>cd cmake-build

>>cmake -DENABLE_AUTOMATIC_INIT_AND_CLEANUP=OFF ..
```

```
>>make
>>make install
```

#### Rapidjson:

```
>>wget https://github.com/miloyip/rapidjson/archive/v1.1.0.tar.gz
>>tar xfvz v1.1.0.tar.gz
>>sudo mv rapidjson-1.1.0/include/rapidjson/ /usr/local/include
```

#### libmicrohttpd:

```
>>wget http://ftp.gnu.org/gnu/libmicrohttpd/libmicrohttpd-0.9.70.tar.gz
>>tar xvf libmicrohttpd-0.9.70.tar.gz
>>cd libmicrohttpd-0.9.70
>>./configure --disable-messages --disable-postprocessor --disable-dauth
>>make
>>sudo make install
>>sudo ldconfig
>>sudo apt install git
```

A continuación, conseguimos ya el código fuente de OCB e instalamos:

```
>>git clone https://github.com/telefonicaid/fiware-orion
>>cd fiware-orion
>>make
>> sudo make install INSTALL_DIR=/usr
>> sudo mkdir /etc/sysconfig #por si no existe
>> sudo cp /opt/fiware-orion/etc/config/contextBroker /etc/sysconfig/
>>sudo touch /var/log/contextBroker/contextBroker.log
>>sudo chown orion /var/log/contextBroker/contextBroker.log
>>sudo cp /opt/fiware-orion/rpm/SOURCES/etc/logrotate.d/logrotate-
contextBroker-daily /etc/logrotate.d/
>>sudo cp /opt/fiware-orion/rpm/SOURCES/etc/sysconfig/logrotate-
contextBroker-size /etc/sysconfig/
>>sudo cp /opt/fiware-orion/rpm/SOURCES/etc/cron.d/cron-logrotate-
contextBroker-size /etc/cron.d/
```

```
>> useradd -M -N orion
```

Por último, comprobamos que se todo haya ido correctamente iniciando el programa(antes iniciamos MongoDB):

```
>> service mongod start  
>> systemctl daemon-reload  
>> systemctl start contextBroker  
>> contextBroker -version  
>> contextBroker -port 1026
```

Una vez ejecutados estos últimos comandos, puede tardar unos minutos en ser totalmente accesible, pero después ya queda arrancado.

Toda la información sobre la instalación se ha obtenido de la página oficial de OCB:



# REFERENCIAS

---

- [1] Lu Tan y Neng Wang, "Future internet: The Internet of Things," 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE), 2010, pp. V5-376-V5-380, doi: 10.1109/ICACTE.2010.5579543.
- [2] Página oficial de FIWARE, "What is FIWARE?", <https://www.fiware.org/about-us/>.
- [3] Sergio García Gómez, Juan José Hierro, Luis Muñoz, Leonor Rodríguez Catalán, Gregorio Ambrosio, Andreu Ibáñez y Paula Llobet, "Fiware: Una plataforma abierta y estándar para Ciudades Inteligentes", en *esmartcity.es*, 13/08/2015.
- [4] Página oficial de FIWARE, "API WALKTHROUGH & DEVELOPMENT INTRO", Documentación oficial de IOT Agent - Json, <https://fiware-iotagent-json.readthedocs.io/en/latest/>.
- [5] Manuel J. Prieto, "Historia de la Criptografía. Cifras, códigos y secretos de la antigua Grecia a la Guerra Fría", Editorial La Esfera De Los Libros, ISBN 978-84-9164-737-9.
- [6] Página oficial de Wireshark, "About Wireshark", <https://www.wireshark.org/>.
- [7] Chih-Chung Lu and Shau-Yin Tseng, "Integrated design of AES (Advanced Encryption Standard) encrypter and decrypter," Proceedings IEEE International Conference on Application- Specific Systems, Architectures, and Processors, 2002, pp. 277-285, doi: 10.1109/ASAP.2002.1030726.
- [8] Leonor Priego García, " Estudio del protocolo TLS (Transport Layer Security)", Proyecto de Fin de Posgrado de la Universitat Oberta de Catalunya.
- [9] Página oficial de MongoDB, "About MongoDB", <https://www.mongodb.com/es>.
- [10] Página oficial de cUrl, " command line tool and library for transferring data with URLs ", <https://curl.se>.
- [11] Página oficial de openssl, "Home", <https://www.openssl.org/>.
- [12] Página oficial de FIWARE, "BUILDING FROM SOURCES ", Documentación oficial de Orion Context Broker, [https://fiware-orion.readthedocs.io/en/master/admin/build\\_source/index.html#ubuntu-2004-lts](https://fiware-orion.readthedocs.io/en/master/admin/build_source/index.html#ubuntu-2004-lts).
- [13] K. Krombholz, K. Busse, K. Pfeffer, M. Smith and E. von Zezschwitz, ""If HTTPS Were Secure, I Wouldn't Need 2FA" - End User and Administrator Mental Models of HTTPS," 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 246-263, doi: 10.1109/SP.2019.00060.
- [14] Página oficial de FIWARE, "SECURITY CONSIDERATIONS", Documentación oficial de Orion Context Broker, <https://fiware-orion.readthedocs.io/en/master/user/security/index.html>.

