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

Resumen

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 workon 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.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

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

En 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 querremos 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 capitulo, 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.

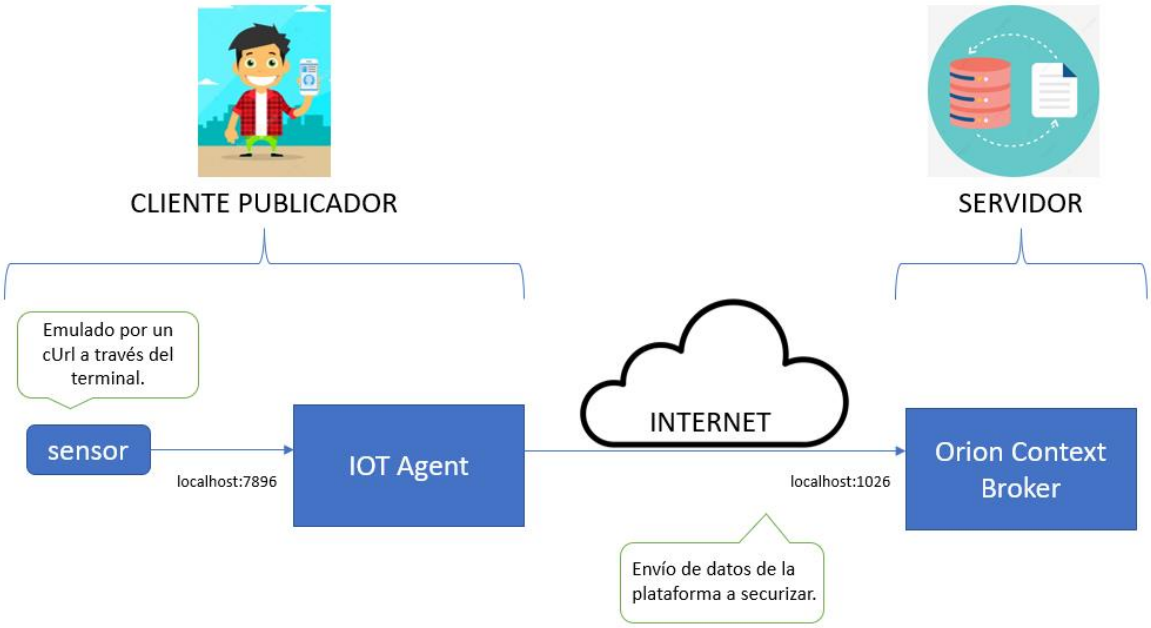

Ilustración 1 – Esquema de la Publicación de datos

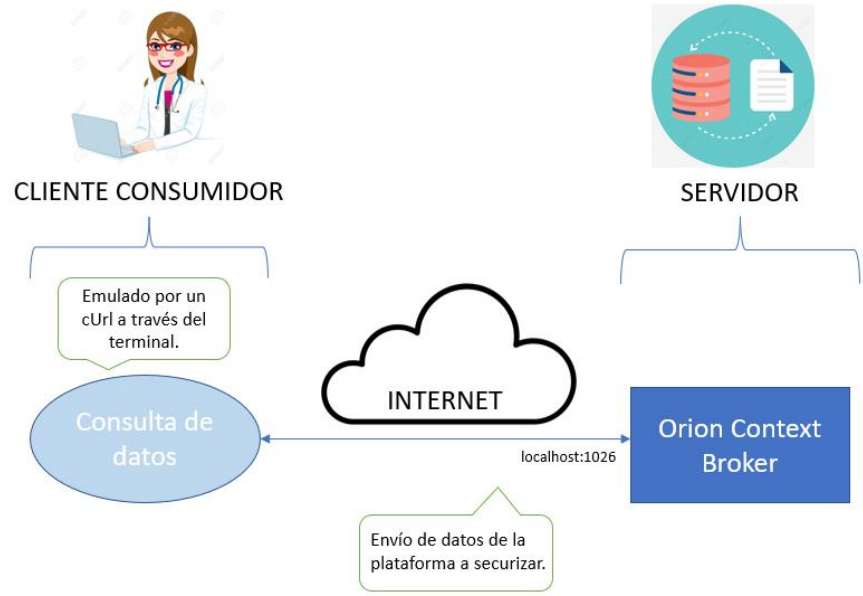

Ilustración 2 – Esquema consumición de datos

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:

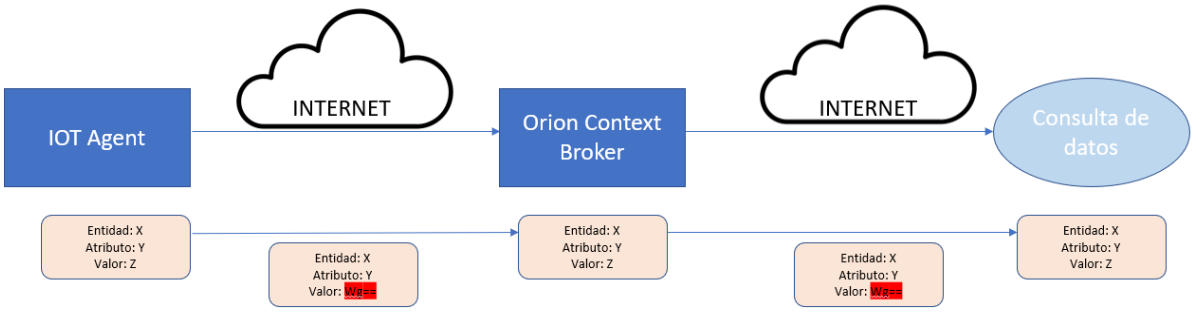

Ilustración 3 – Esquema del escenario de encriptación AES

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

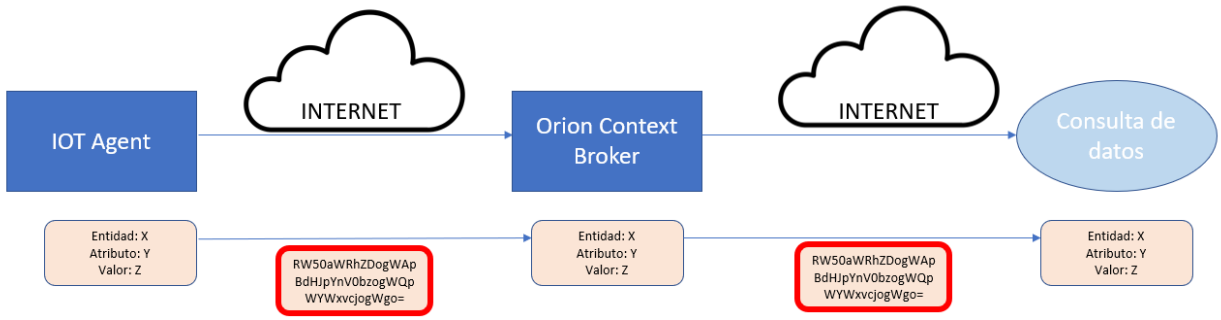

Ilustración 4 – Esquema del escenario de encriptación HTTPS

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
Total	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
Total	15h	40h

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

Subtarea	**Tiempo estimado**	**Tiempo real**
Encriptación en OCB	5h	15h
Encriptación en IOTA	5h	10h
Pruebas	25h	25h
Total	35h	50h

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

Subtarea	**Tiempo estimado**	**Tiempo real**
Planteamiento escenarios	2h	2h
Pruebas de escenarios	5h	5h
Total	7h	7h

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

Subtarea	**Tiempo estimado**	**Tiempo real**
Memoria del trabajo	90h	90h
Total	90h	90h

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 autovías 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 limite 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 plaforma 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.

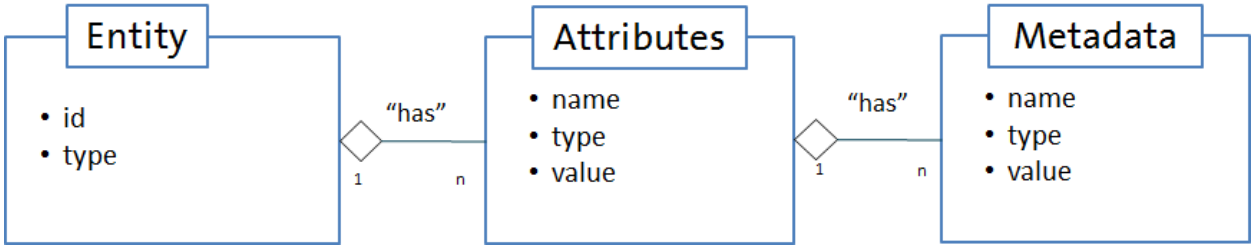

Ilustración 7 - Esquema de contextos de información basados en entidades

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].

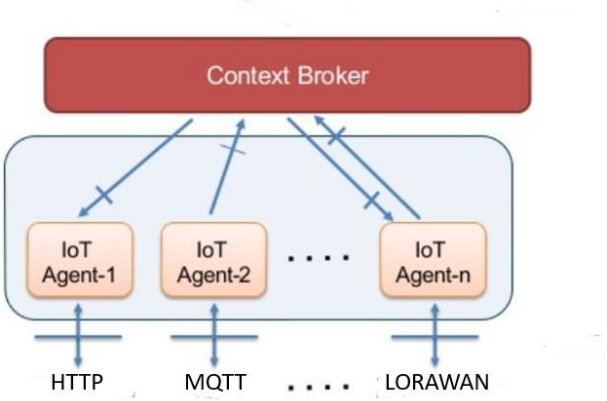

Ilustración 8-Esquema de interconexión del IOT Agent

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 asimetrico

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 desencriptación. 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 tranmisión, hasta que un método análogo inverso la descifre en el receptor. Ejemplos de éste pueden ser DES, o su evolución, AES.

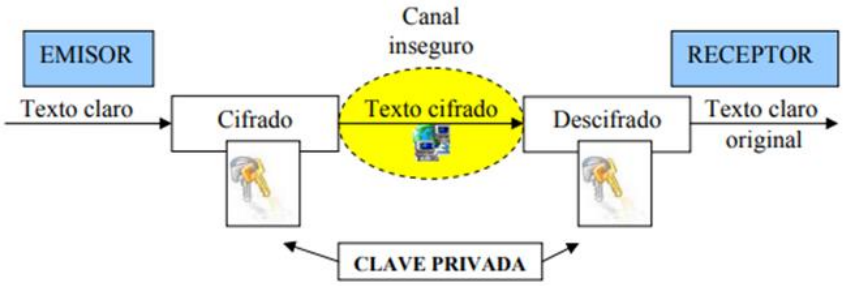

Ilustración 9 - Esquema de funcionamiento de cifrado simétrico

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.

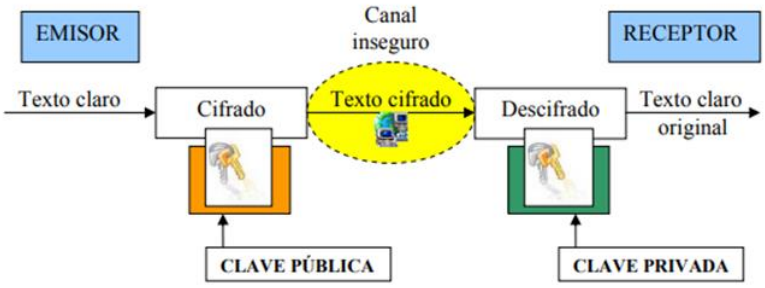

Ilustración 10 - Esquema de funcionamiento de cifrado asimétrico

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, RC6TM, 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 desencriptar el mensaje.

El algoritmo de Rindjael está hecho con la intención de que, si se intenta desencriptar 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 operaciónes 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 desencriptación, 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
| MixColumes	| InvAddRoundKey
| AddRoundKey	| InvMixColumes
-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 desencriptado 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 informacion 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 desencriptar con la clave privada de ese mismo usuario.

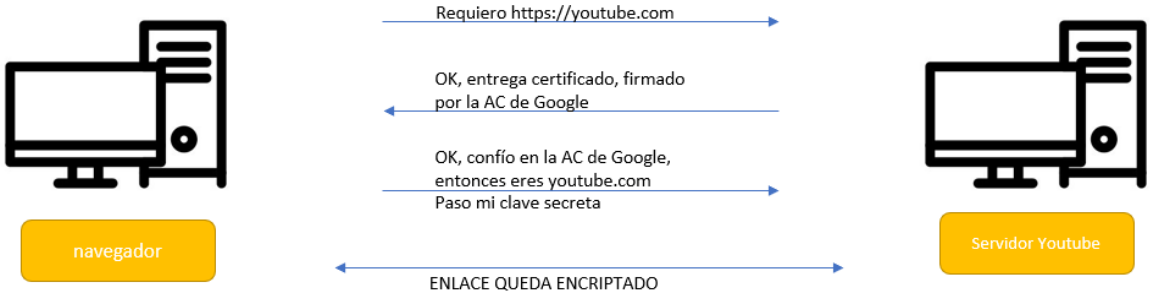

Ilustración 11 - Esquema de ejemplo de uso de HTTPS

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.

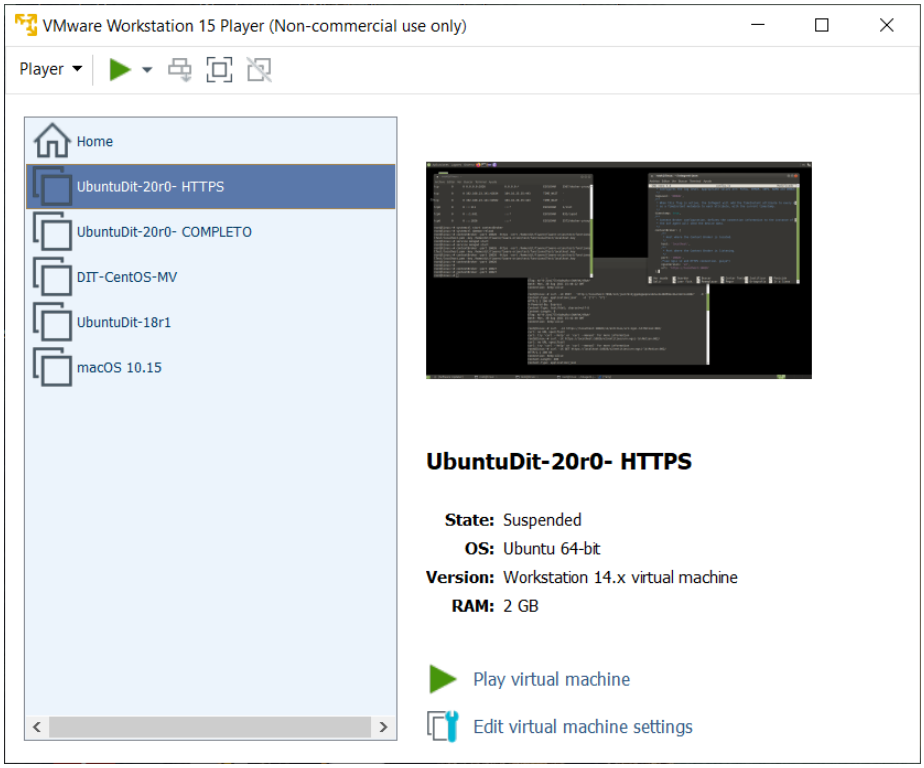

Ilustración 12 -Vmplayer Workstation corriendo máquina virtual Ubuntu

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 datos y archivos por internet. *cUrl* es usada en línea de commandos 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 intererese [10].

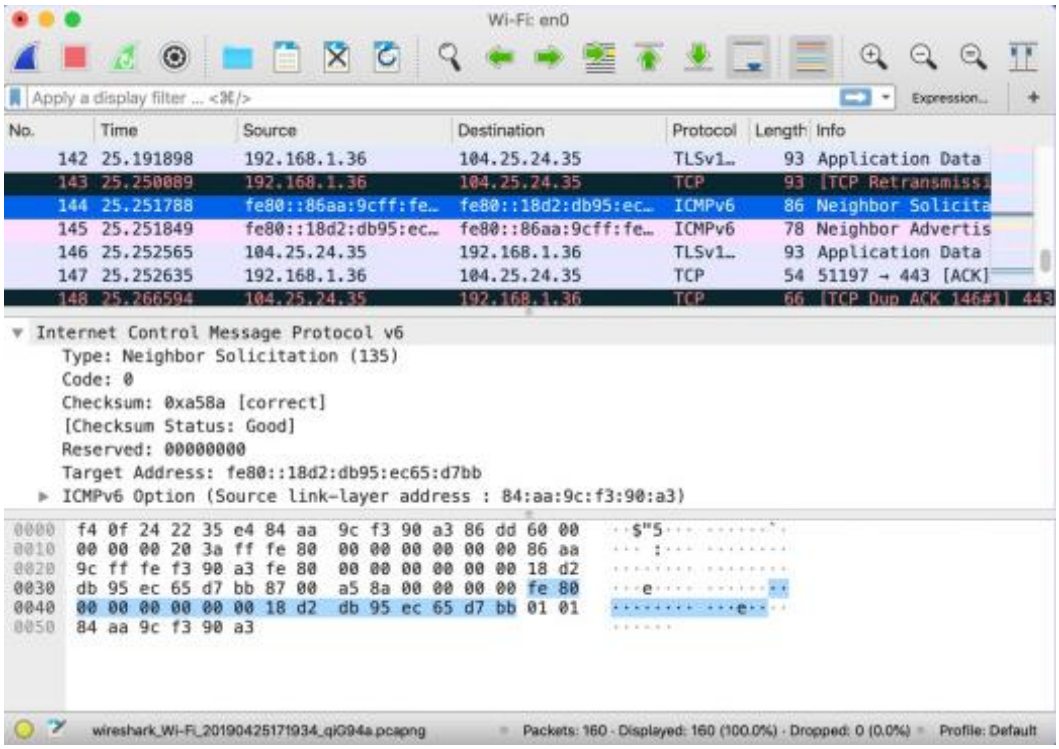

Ilustración 15 - Ejemplo de captura de paquetes del programa Wireshark

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 danto 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.

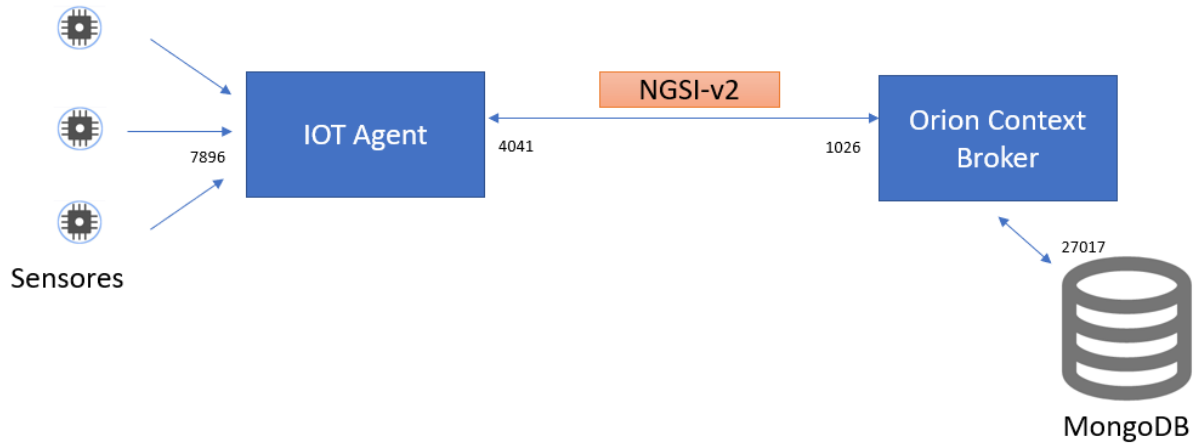

Ilustración 17 - Esquema de la plataforma IOT basada en FIWARE que se va a implementar

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 desarollada 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'.

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 compobación 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.

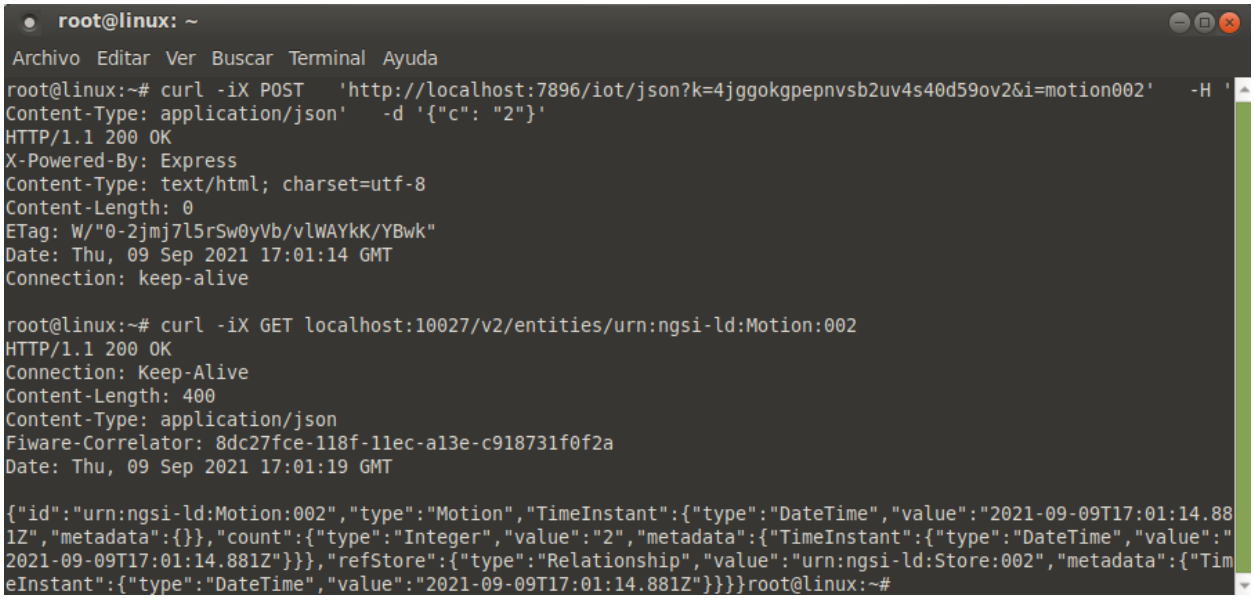

Ilustración 19 - Comprobación del arranque y funcionamiento de IOT AGENT

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 asi 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.

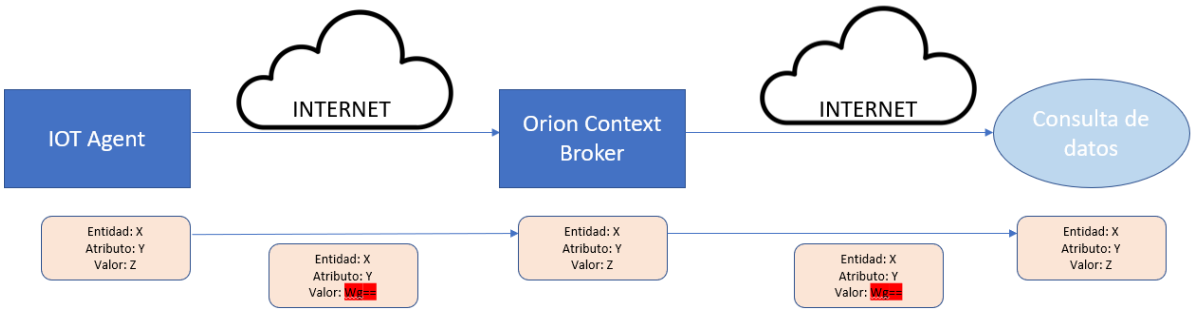

Ilustración 20-Esquema del escenario 1

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 ésto 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.

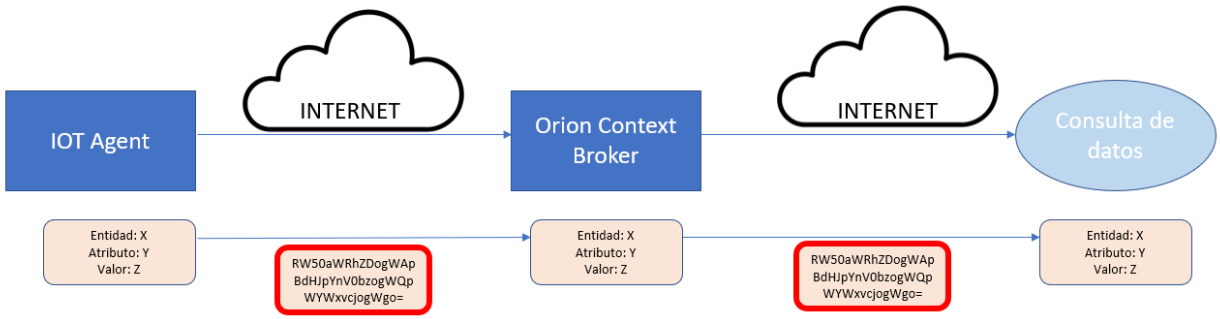

Ilustración 21- Escquema de descripción del escenario 2

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 cuál 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.

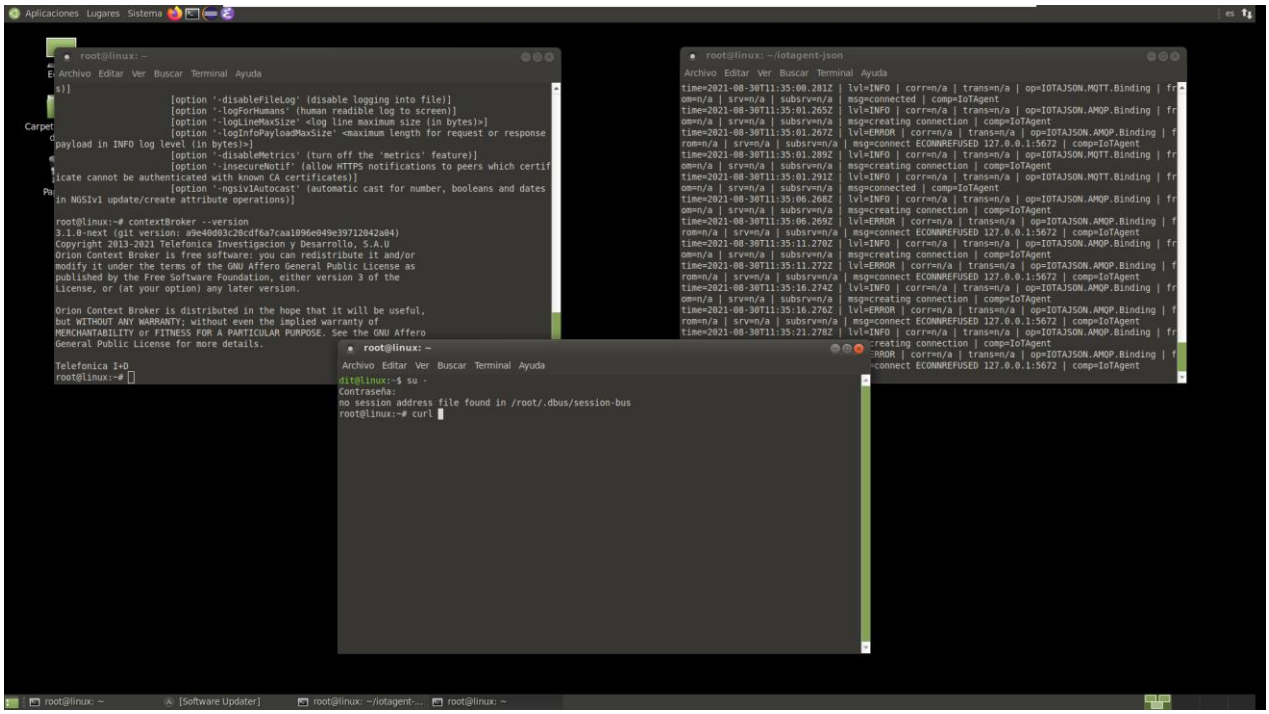

Ilustración 22- Simulación del escenario

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.

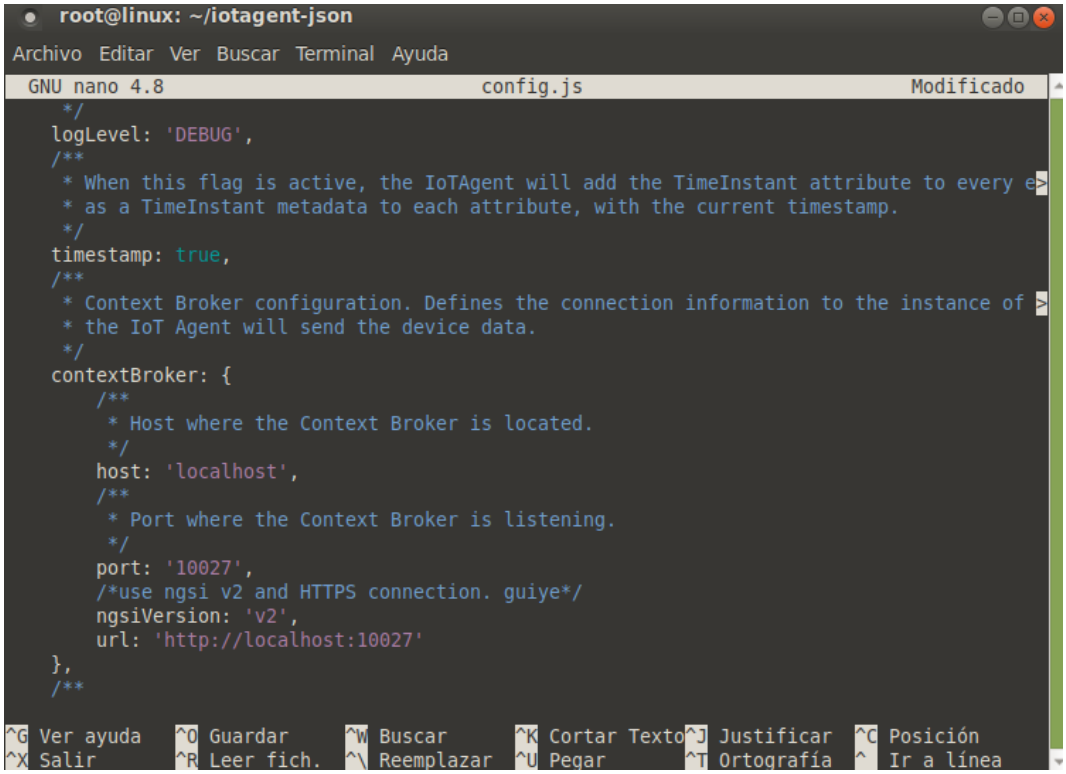

Ilustración 23 - Fichero de configuración de IOTA para el caso HTTP

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:

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.

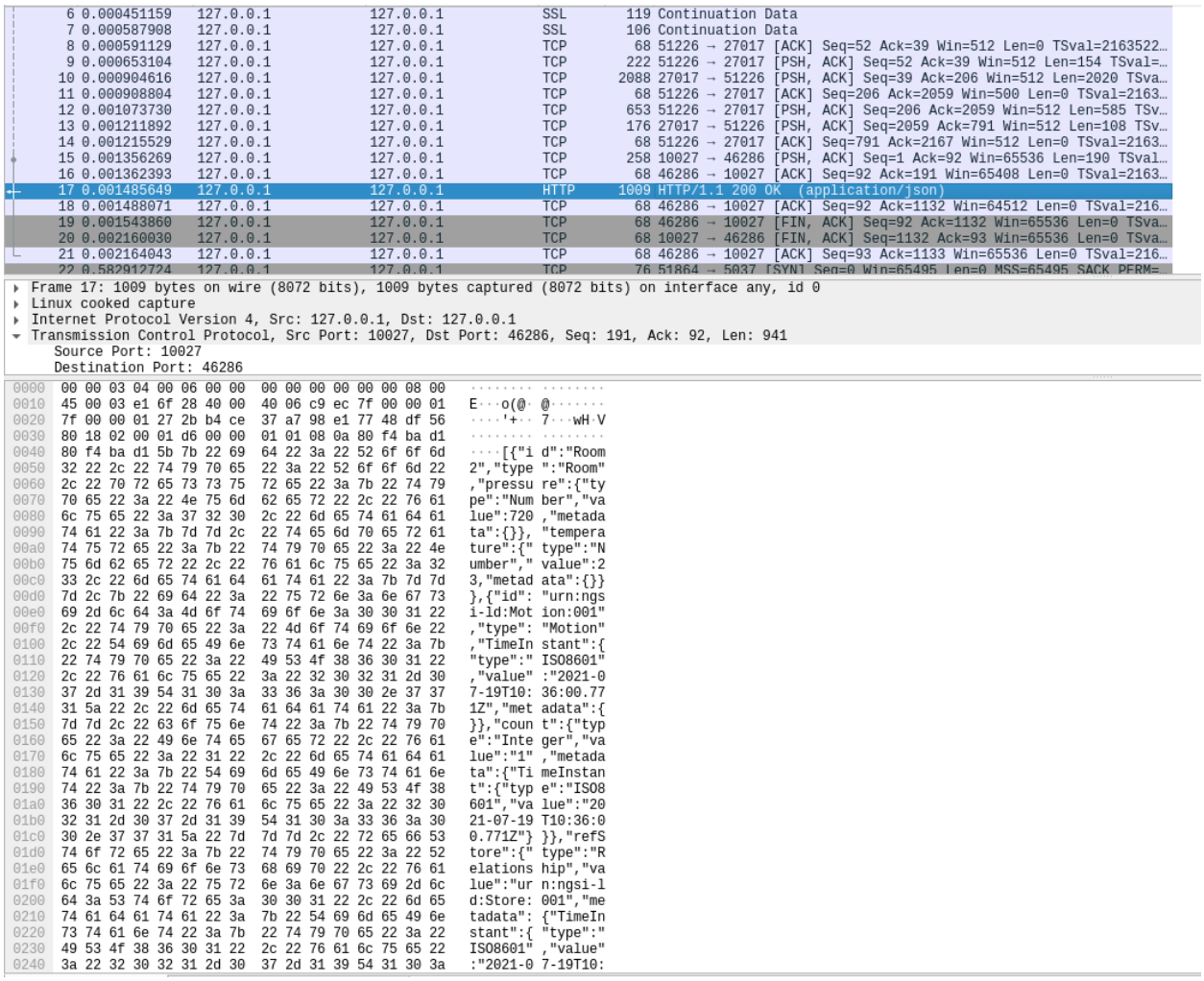

Ilustración 25 - Petición de consulta captada por Wireshark

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=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: 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.

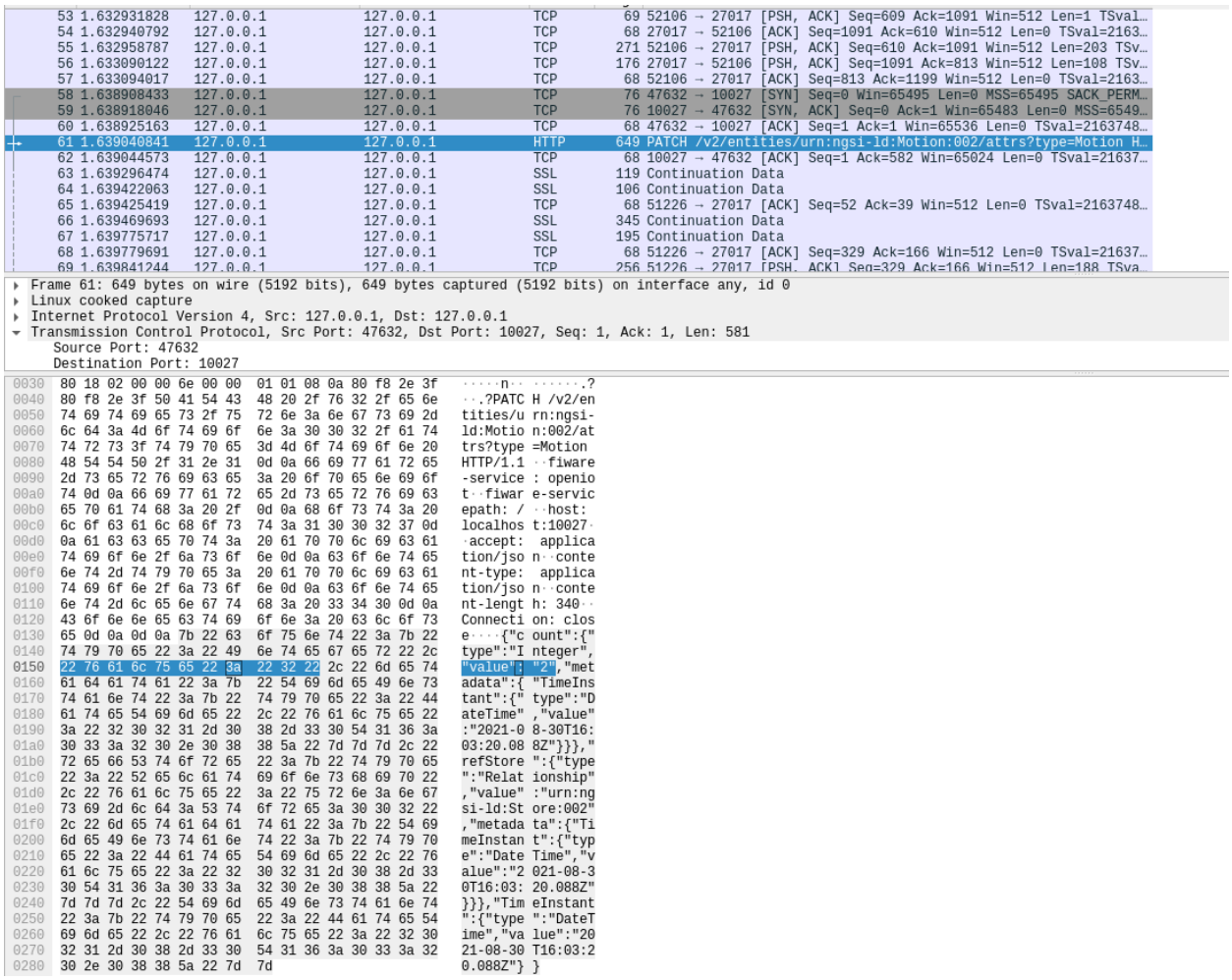

Ilustración 27-Captura del paquete que envía el valor que actualiza el sensor

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.

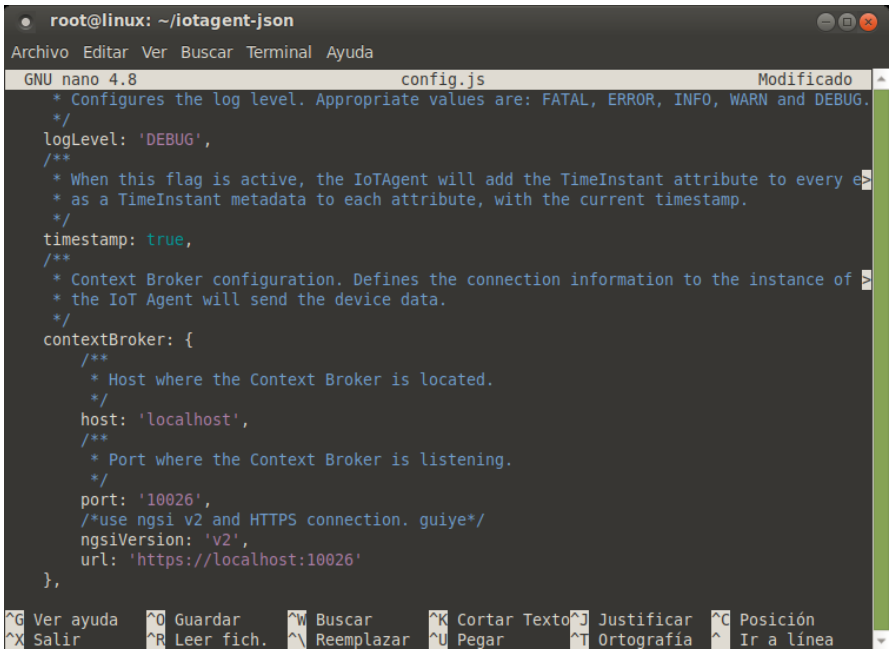

Ilustración 28 - Modificación del fichero de configuración de IOTA para HTTPS

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":"2021-08-30T15:42:40.10
4Z","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":{"Tim
eInstant":{"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.

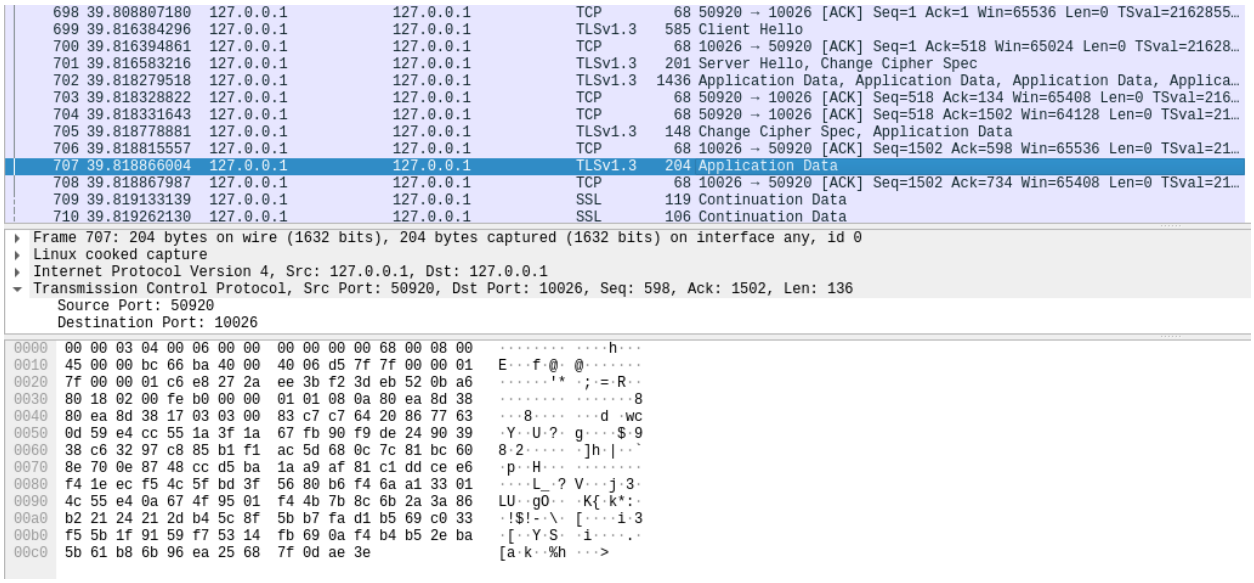

Ilustración 30 - Captura mediante Wireshark de un paquete de intercambio de información en la Consulta a OCB

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.

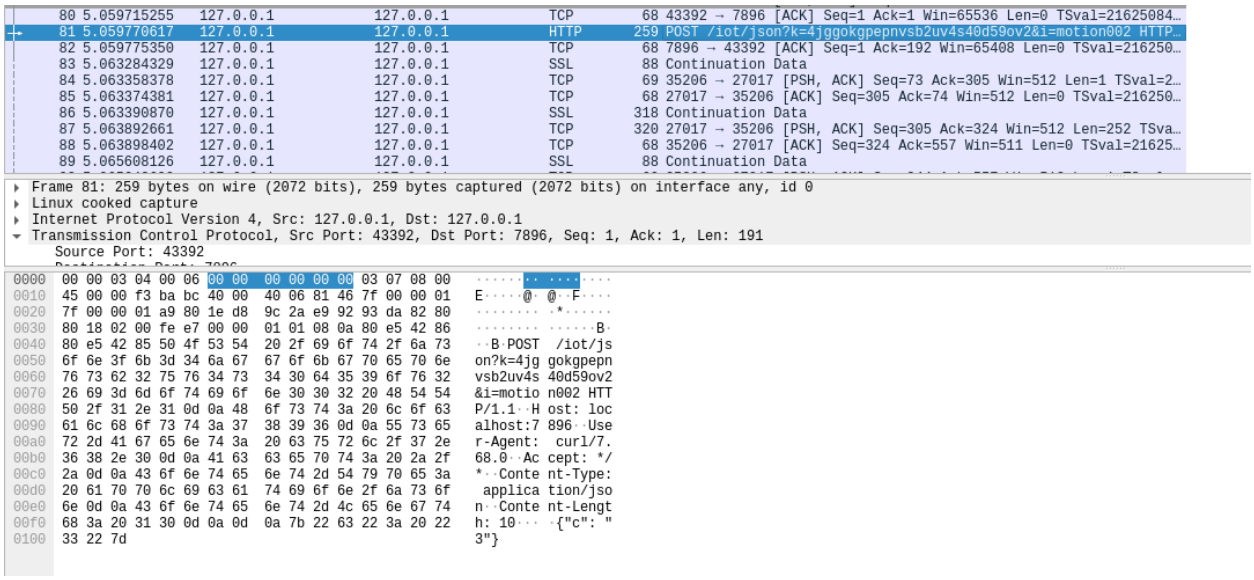

Ilustración 32 - Petición que simula al sensor capturada mediante Wireshark

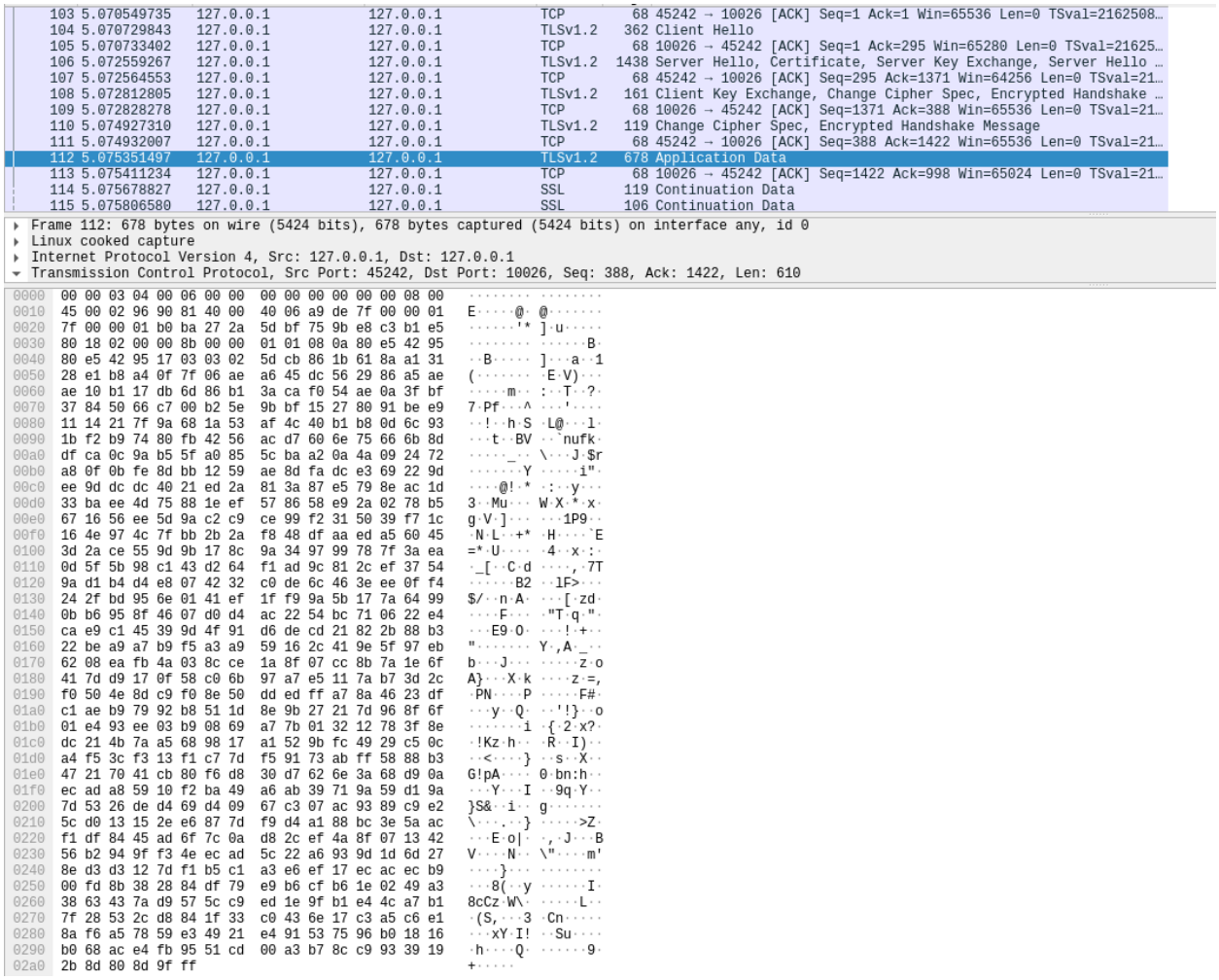

Ilustración 33 - Comunicación entre IOTA y OCB, capturada mediante Wireshark

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.

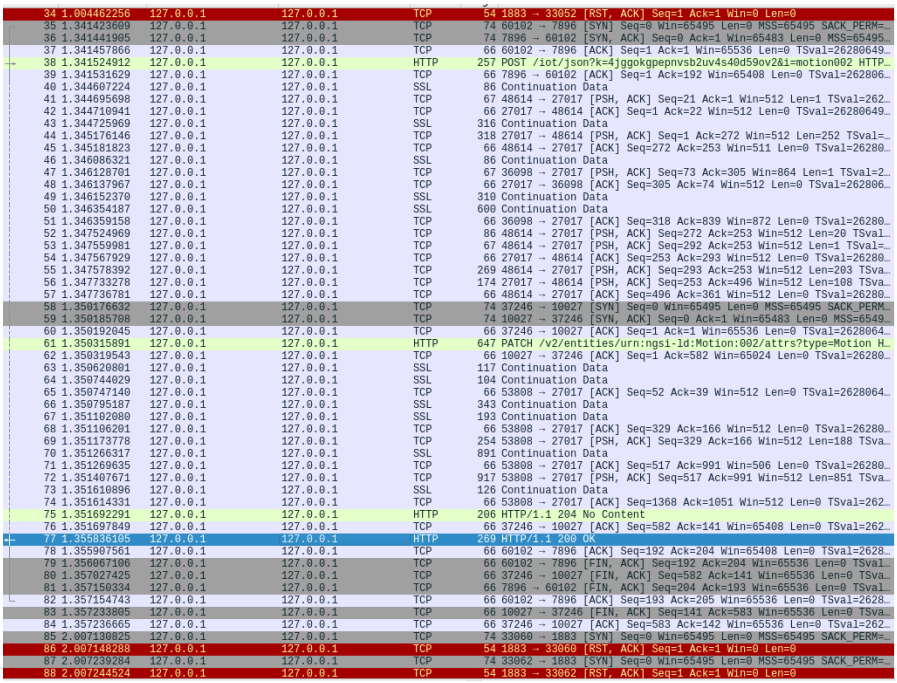

Ilustración 34 - Captura de paquetes mediante Wireshark del proceso de actualización de contexto sin encriptar

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.

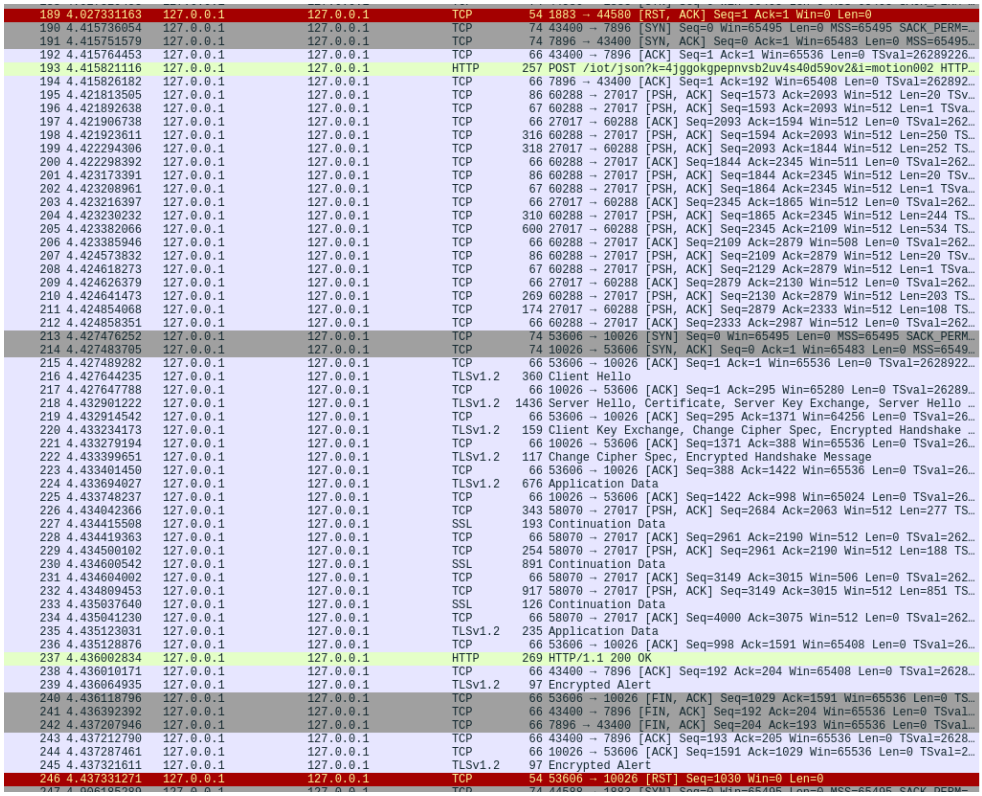

Ilustración 35 - Captura de paquetes mediante Wireshark del proceso de actualización de contexto encriptado mediante HTTPS

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 alejadada 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 mas 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 confíe 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      \
           libgcrypt-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 codigo 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.*

[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.*