# Introducción

Las redes de Internet de las Cosas (IoT) pueden ser implementadas utilizando diversos protocolos de comunicaciones. Los más habituales al crear estas redes son: HTTP, CoAP, MQTT y XMPP, y en la selección de estos protocolos, dejamos de lado los medios de comunicación que, exclusivamente, son M2M (Machine to Machine) debido a que estos últimos, por su definición de comunicación exclusivamente entre maquinas, no entran en la categoría de protocolo IoT, propiamente tal. En la Tabla 1 se comparan los protocolos IoT antes mencionados, se expone las diversas características discriminatorias al momento de realizar la elección del mejor protocolo, según la necesidad.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Característica** | **HTTP** | **CoAP** | **MQTT** | **XMPP** |
|  | **[7]** | **[15]** | **[1]** | **[10] [14]** |
| Request/  Response | ✓ | ✓ | ✗ | ✓ |
| Publish/  Subscribe | ✗ | ✗ | ✓ | ✓ |
| Multicast | ✗ | ✓ | ✗ | ✓ |
| Events or  Push | ✓ | ✓ | ✓ | ✓ |
| Bypasses firewall | ✗ | (✓) | ✓ | ✓ |
| Federation | ✗ | ✗ | ✗ | ✓✓ |
| Authentication | ✓ | ✓ | ✓ | ✓ |
| Network Identity | (✓) | (✓) | ✗ | ✓ |
| Authorization | ✗ | ✗ | ✗ | ✓✓ |
| Encryption | ✓ | ✓ | ✓ | ✓ |
| End-to-end encryption | ✗ | ✗ | ✗ | ✓✓ |
| Compression | ✓ | ✗ | ✗ | ✓ |
| Streaming | ✓ | ✗ | ✓ | ✓ |
| Reliable messaging | ✗ | ✗ | ✓✓ | ✗ |
| Message  Queues | ✗ | ✗ | ✗ | ✗ |

Table 1: Comparativa protocolos IoT más utilizados, donde el símbolo «✗» quiere decir: no cumple la característica. El signo «✓»: cumple parcialmente. «(✓)»: Lo hace parcialmente. Y «✓✓»: Hay más de un componente que realiza dicha función.

En cada uno de los protocolos listados en la Tabla 1 existen desafíos en el área de seguridad. Por ejemplo, con HTTP se pueden crear sistemas altamente robustos y seguros, sin embargo, el problema de la topología de HTTP es un gran obstáculo. Debido a que los clientes se pueden conectar a los servidores, pero normalmente los servidores no se pueden volver a conectar a los clientes, ya que la mayoría de ellos se encuentran detrás de los firewalls.

CoAP [15], es un protocolo de transferencia RESTful para nodos y redes con restricciones. Es escalable, tiene la posibilidad de implementar Datagram Transport Layer Security (DTLS) [9], sin embargo, aún así posee una pobre capa de seguridad y el Bypass de Firewall es bastante limitado, lo cual hace que exponer servicios a través de Internet sea bastante engorroso, monetariamente costoso y complejo. Por otro lado, el protocolo Message Queuing Telemetry Transport (MQTT) está basado en el patrón de diseño Publish-Suscribe, a diferencia de CoAP realiza un eficaz bypass de los firewalls, y soporta encriptación SSL/TLS, características realmente importantes cuando hablamos de redes IoT, sin embargo, el protocolo MQTT tiene serias vulnerabilidades conocidas [11], a tal nivel de gravedad, que, incluso gobiernos han llamado a no usarlo y han prohibido su uso en sistemas gubernamentales.

Al final, esta listado el Protocolo extensible de mensajería y comunicación de presencia (XMPP - por sus siglas en inglés) [10] [14], este es un protocolo abierto basado en flujos XML para la comunicación en tiempo real. XMPP fue desarrollado en el año 1998 con el nombre Jabber, después estandarizado por la Internet Engineering Task Force (IETF) liberando así su primera versión oficial bajo el nombre de Jabber, para luego, en el año 2002, pasar a conocerse con el nombre XMPP. El protocolo es estandarizado por Fundación de Estándares XMPP (XSF - por sus siglas en ingles), quienes corrigen errores y extienden las funcionalidades de este protocolo, por medio de los que se conocen por Protocolos de Extension de XMPP (XEPs) [12]. Inicialmente fue desarrollado para mensajería instantánea, pero con el paso del tiempo, ha sido utilizado ampliamente para diversos fines iniciando desde IM, como para videojuegos, notificaciones Push, Base de datos descentralizadas, redes sociales, microblogging, sistemas de videoconferencias, y finalmente Internet de las Cosas. Es un protocolo muy versátil, probado, con una larga data, y escalable. Esto, sumado al soporte de XSF, es lo hace que XMPP sea un protocolo actualizado y seguro.

Como podemos apreciar en la Tabla 1, XMPP es el que cumple de mejor manera las características listadas para seleccionar un protocolo IoT con difusión en la red de Internet.

A pesar de lo mencionado, existe un problema habitual al momento de crear una red XMPP-IoT, que reside en brindar una experiencia transparente para el usuario final, el cual desea adicionar nuevas “cosas” (tipo Plug and Play), tales como sensores, actuadores, controladores o concentradores a la red. Lamentablemente, para este, el uso interoperable de Things de diversos fabricantes de dispositivos, no es viable, ya que cada uno de estos, utiliza protocolos distintos o redes cerradas. Para solucionar esta problemática existe en XMPP, el registro en banda, mecanismo descrito en el XEP-0077: In-Band Registration, el que permite la creación de nuevas identidades donde cada una representa un dispositivo - utilizando el mismo canal de comunicaciones para conexión a los servidores de la red XMPP-IoT, de este modo, cada fabricante puede crear identidades para sus dispositivos en la red, simplificando el control unificado de estos. El problema en este caso es que no hay control sobre cuantas identidades crea el fabricante, ni mucho menos un límite en la cantidad del registro de nuevas cuentas para los dispositivos de este último.

En efecto, es útil cuando hablamos de redes XMPP-IoT de un sólo fabricante. En este caso, la gran mayoría de los servidores, tienen implementado el XEP-0077, para realizar el registro en banda, usando el mismo protocolo XMPP, así se evita usar el método alternativo de crear una web API para crear nuevas identidades, añadiendo posibles vulnerabilidades extra a nuestra red y reduciendo la flexibilidad de esta. El problema fundamental con el Registro en banda, propiamente tal, es que deja totalmente expuesta nuestra red XMPP-IoT al ataque de usuarios/bots maliciosos, ya que pueden sobrecargar nuestros servidores, con solicitudes virtualmente ilimitadas de creación de nuevas cuentas, y los servidores no tienen la capacidad de discriminar las solicitudes benignas de las nocivas. Frente a tal desafío, en los inicios de XMPP, se desarrolló el XEP-0158: CAPTCHA Forms, que describe el mecanismo que opera en conjunto del registro en-banda, para añadirle CAPTCHAs a este último. Esta solución fue sumamente útil para proteger las redes XMPP de los ataques de bots maliciosos durante los primeros años de la popularidad de la mensajería instantánea, cuando el poder de cómputo de los ordenadores personales era bastante inferior al de la actualidad. Pero hoy, no es suficiente, debido a que por medio de visión computacional es relativamente sencillo resolver los CAPTCHAs [3].

De todos modos, cuando se utiliza el XEP-0077 en conjunto de CAPTCHAS Forms para filtrar a los usuarios maliciosos, las “cosas” - que en estricto rigor son bots - no pueden registrarse de forma automatizada en la red XMPP-IoT. Así que esta opción, si fuese segura, sabiendo que no lo es, no es una opción viable para crear dispositivos “plug and play”. Para este fin, es decir, para crear una red XMPP-IoT con la capacidad de registrar en-banda nuevas identidades, generalmente se activa el registro en banda, sin habilitar CAPTCHAS Forms. Lo cual, nos permite tener dispositivos “plug and play”. Esto, al mismo tiempo, abre una brecha en la seguridad de la red XMPP-IoT. Permitiendo que cualquiera - cliente inocuo o maligno, bot benigno o malicioso - pueda visualizar a nivel de red el servidor y esté plenamente facultado para registrar identidades en el servidor, lo que abre una brecha en la seguridad en la red XMPP-IoT.

Este trabajo posee las siguientes secciones. Sección 2 presenta la propuesta para disminuir los ataques de red y dar una alternativa de interoperabilidad a los fabricantes de dispositivos. Sección 3 describe como esta propuesta es implementada en un servidor y cliente XMPP para verificar con un experimento que efectivamente los ataques se reducen a cero. Sección 4 describe las conclusiones y trabajos futuros.

# Propuesta

Como ya hemos visto en la Tabla 1, XMPP es más robusto en términos de seguridad que el resto de los protocolos comparados, ya que existen XEPs que tributan en esta área. Tal como lo es el protocolo de extensión de XMPP XEP-0348: Signing Forms[16], que describe el mecanismo para realizar el firmado de formularios, con credenciales que no tienen directa relación con la conexión - usando el algoritmo modificado de OAuth 1.0[8] para así concretar el registro de nuevas identidades en la red XMPP, esto es, usando el protocolo de extensión XEP-0077[13] que describe el mecanismo para realizar un registro en banda de nuevas cuentas en el servidor XMPP.

En términos prácticos, al unificar estas dos tecnologías, es decir, el registro en banda junto al proceso de firmado de formularios de registro se dota a la red XMPP-IoT de la capacidad de otorgar el permiso para registrar nuevas identidades a los fabricantes de dispositivos IoT (Cosas). Donde las credenciales para crear estas nuevas identidades en la red, no tienen relación con la conexión o ingreso a la red XMPP, estas toman el nombre de credenciales de consumidor que es el par “consumer key” y “consumer secret”, las que a su vez se asocian a un cantidad de registro de nuevas identidades permitidas en la red por dicha credencial, como por ejemplo: la consumer key=00000000 perteneciente al manufacturador de dispositivos BinaryLamp, sólo podrá crear 1000 identidades en el servidor XMPP-IoT, posibilitando, de este modo, la interoperabilidad de dispositivos IoT de distintas compañías, sacando provecho de la federalización de XMPP o mediante la concesión de credenciales de consumidor a los fabricantes de dispositivos IoT, para que estos puedan registrar nuevas identidades en la red, por medio de estas credenciales o dos opciones anteriores en su conjunto.

Para dar solución a la problemática expuesta, debemos seleccionar un servidor XMPP. En la actualidad existe una gran cantidad de servidores XMPP. Existen de uso libre o de pago, de código abierto o propietario y desarrollado en diversos lenguajes de programación, documentación variada, con una comunidad existente en torno al servidor, que operan sobre diferentes sistemas operativos y con diversas XEPs implementadas. *IoT Broker* (https://waher.se/Broker.md), servidor XMPP de uso no libre, no es open-source y no hay una comunidad incipiente en torno al proyecto. *AstraChat Isode M-Link*, cuyo uso es de pago, mientras que *ejabberd*, *Tigase* y *Openfire* son de uso libre. Los primeros dos poseen una comunidad también incipiente en torno al desarrollo de nuevas características. Por otro lado, tenemos *Openfire* que es de uso libre, open-source, esta codificado en lenguaje Java, funciona bajo los Sistemas Operativos Linux, macOS, Solaris y Windows, inicialmente desarrollado por la empresa Jive Software, quienes ayudaron ayudador a formar la gran comunidad que mantiene activo el proyecto en la actualidad, llamada Igniterealtime. La misma comunidad posee *SMACK*, una librería que tiene una potente API para crear clientes XMPP, la que ostenta las mismas características mencionadas del servidor.

El servidor XMPP Openfire incluye soporte completo de RFC XMPP, así como las extensiones más comunes. La Tabla 2 a continuación detalla el nivel de soporte para los requisitos establecidos por XEP-0302: XMPP Compliance Suites 2012.

|  |  |
| --- | --- |
| **Especificación** | **Soportado** |
| RFC 6120: XMPP Core | Sí |
| RFC 6121: XMPP IM | Sí |
| RFC 7622: XMPP ADDR | Sí |
| XEP-0030: Service Discovery | Sí |
| XEP-0114: Jabber Component Protocol | Sí |

Table 2: Tabla: Openfire Soporte XMPP Core

El cumplimiento de soporte de nivel avanzado incluye el conjunto básico completo, así como características más avanzadas de uso común para los clientes XMPP. La Tabla 3 detalla el nivel de soporte para los requisitos establecidos por XMPP Compliance Suites 2012.

|  |  |
| --- | --- |
| **Especificación** | **Soportado** |
| XEP-0115: Entity Capabilities | Sí |
| XEP-0191: Blocking Command | No |
| XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH) | Sí |
| XEP-0206: XMPP Over BOSH | Sí |
| XEP-0054: vcard-temp | Sí |
| XEP-0163: Personal Eventing Protocol | Sí |
| XEP-0045: Multi-User Chat | Sí |
| XEP-0198: Stream Management | Parcialmente |

Table 3: Tabla: Openfire soporte avanzado

En la Tabla 4 se listan los XEP compatibles con Openfire. Los XEP que solo requieren soporte del lado del cliente se omiten.

Para crear el cliente usamos **SMACK**, el cual soporta las siguientes XEPs, listadas en la Tabla 5 y Tabla 6.

|  |
| --- |
| **Especificación** |
| XEP-0004: Data Forms |
| XEP-0012: Last Activity |
| XEP-0013: Flexible Offline Message Retrieval |
| XEP-0030: Service Discovery |
| XEP-0033: Extended Stanza Addressing |
| XEP-0049: Private XML Storage |
| XEP-0050: Ad-Hoc Commands |
| XEP-0054: vcard-temp |
| XEP-0055: Jabber Search |
| XEP-0059: Result Set Management |
| XEP-0060: Publish-Subscribe |
| XEP-0065: SOCKS5 Bytestreams |
| XEP-0077: In-Band Registration |
| XEP-0078: Non-SASL Authentication |
| XEP-0082: XMPP Date and Time Profiles |
| XEP-0086: Error Condition Mappings |
| XEP-0092: Software Version |
| XEP-0096: File Transfer |
| XEP-0106: JID Escaping |
| XEP-0114: Jabber Component Protocol |
| XEP-0115: Entity Capabilities |
| XEP-0124: HTTP Binding |
| XEP-0126: Invisibility |
| XEP-0128: Service Discovery Extensions |
| XEP-0138: Stream Compression |
| XEP-0160: Best Practices for Handling Offline Messages |
| XEP-0163: Personal Eventing via Pubsub |
| XEP-0198: Stream Management (parcial) |
| XEP-0202: Entity Time |
| XEP-0203: Delayed Delivery |
| XEP-0280: Message Carbons |

Table 4: XEPs soportadas por Openfire

|  |  |  |
| --- | --- | --- |
| Especificación |  | Especificación |
| n/a-Google GCM JSON payload | XEP-0138-Stream Compression |
| n/a-Group Chat Invitations | XEP-0141-Data Forms Layout |
| n/a-Jive Properties | XEP-0163-Personal Eventing Protocol |
| XEP-0004-Data Forms | XEP-0184-Message Delivery Receipts |
| XEP-0012-Last Activity | XEP-0191-Blocking Command |
| XEP-0013-Flexible Offline Message Retrieval | XEP-0199-XMPP Ping |
| XEP-0016-Privacy Lists | XEP-0202-Entity Time |
| XEP-0022-Message Events | XEP-0203-Delayed Delivery |
| XEP-0030-Service Discovery | XEP-0206-XMPP Over BOSH |
| XEP-0033-Extended Stanza Addressing | XEP-0224-Attention |
| XEP-0045-Multi User Chat | XEP-0231-Bits of Binary |
| XEP-0047-In-Band Bytestreams | XEP-0280-Message Carbons |
| XEP-0048-Bookmarks | XEP-0296-Best Practices for Resource Locking |
| XEP-0049-Private Data | XEP-0308-Last Message Correction |
| XEP-0050-Ad-Hoc Commands | XEP-0313-Message Archive Management |
| XEP-0054-vcard-temp | XEP-0319-Last User Interaction in Presence |
| XEP-0055-Jabber Search | XEP-0323-Internet of Things - Sensor Data |
| XEP-0059-Result Set Management | XEP-0324-Internet of Things - Provisioning |
| XEP-0060-PubSub | XEP-0325-Internet of Things - Control |
| XEP-0065-SOCKS5 Bytestreams | XEP-0332-HTTP over XMPP transport |
| XEP-0071-XHTML-IM | XEP-0333-Chat Markers |
| XEP-0077-In-Band Registration | XEP-0334-Message Processing Hints |
| XEP-0079-Advanced Message Processing | XEP-0335-JSON Containers |
| XEP-0080-User Location | XEP-0347-Internet of Things - Discovery |
| XEP-0082-XMPP Date Time Profiles | XEP-0352-Client State Indication |
| XEP-0085-Chat State Notifications | XEP-0357-Push Notifications |
| XEP-0090-Time Exchange | XEP-0359-Stable and Unique Stanza IDs |
| XEP-0092-Software Version | XEP-0363-HTTP File Upload |
| XEP-0093-Roster Item Exchange | XEP-0372-References |
| XEP-0095-Stream Initation | XEP-0382-Spoiler Messages |
| XEP-0096-SI File Transfer | XEP-0384-OMEMO Multi End  Message and Object Encryption |
| XEP-0115-Entity Capabilities |
| XEP-0116-Jingle | XEP-0392-Consistent Color Generation |
| XEP-0122-Data Forms Validation | XEP-0394-Message Markup |
| XEP-0133-Service Administration | XEP-xxxx-Multi-User Chat Light |

Table 5: XEPs soportadas por librería SMACK Table 6: XEPs soportadas por librería SMACK En este trabajo proponemos desarrollar e implementar el mecanismo descrito en la especificación **XEP-0348: Signing Forms** sobre la **XEP-0077: In-band Registration**, tanto en el servidor como en el cliente, para que, de este modo, se pueda asignar a un cierto lote definido de clientes la facultad de crear identidades en la red XMPP-IoT. Con esto reduciremos a cero la creación de identidades por usuarios o bot maliciosos mediante la vulnerabilidad de la XEP-0077. En particular, para validar nuestra propuesta, realizaremos un experimento implementándola en el cliente *SMACK* y en el servidor *Openfire* tal como se aprecia en las Figuras 1 y 2.

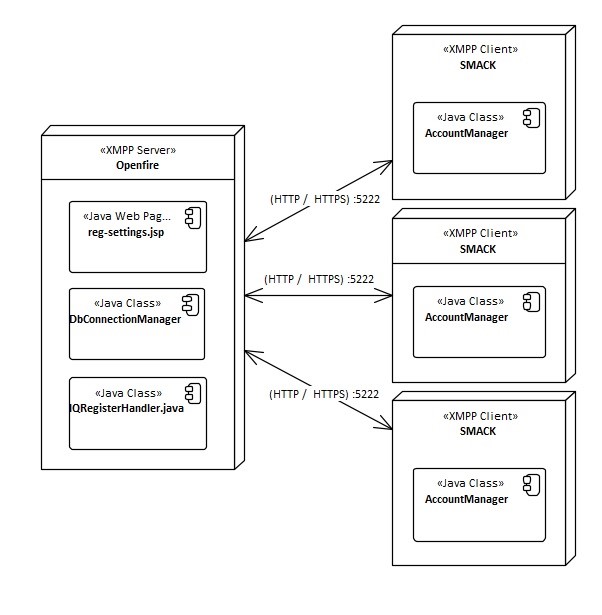


Figure 1: Diagrama de despliegue

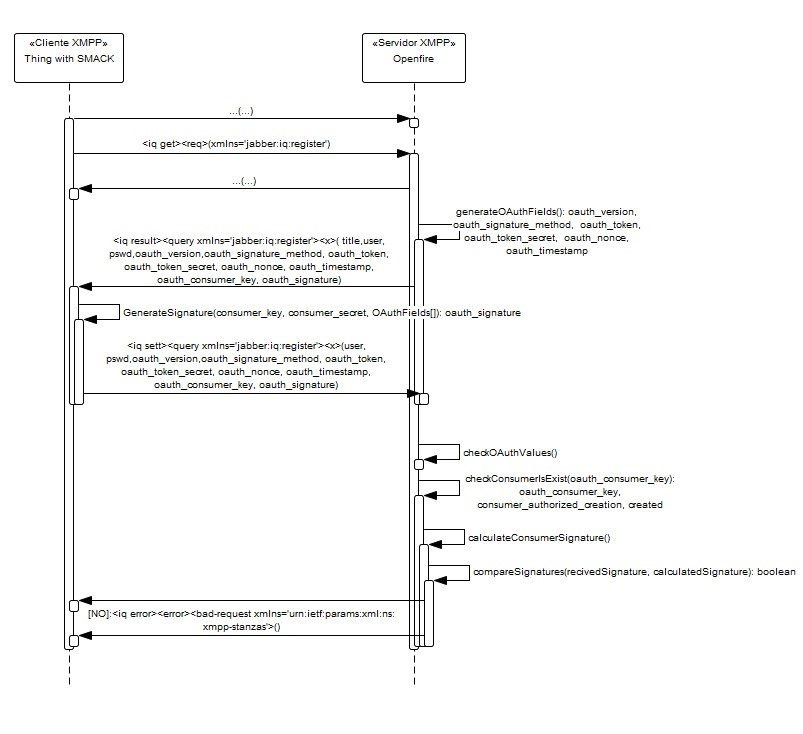


Figure 2: Diagrama de secuencia de Signing Forms

Para ello se planificó el desarrollo de este nuevo mecanismo en el servidor XMPP Openfire y en la librería SMACK, desarrollo que constó de cuatro liberaciones relevantes:

1. Bot que demuestra la vulnerabilidad existente en la XEP-0077 en el contexto de IoT.
2. Servidor Openfire con las mejoras para incorporar el XEP-0348.
3. Librería SMACK incorporando mejoras para implementando el XEP-0348.
4. Cliente Thing que utiliza la versión mejorada de SMACK, para validar la propuesta.

Inicialmente en el desarrollo, se codifico un pequeño bot, utilizando la librería SMACK, el cual tiene la facultad de crear, virtualmente, ilimitadas identidades en el servidor Openfire. Los resultados de este se pueden ver en la Sección 3. Después de la creación del bot, se continuo con el análisis, diseño, codificación y con las respectivas pruebas de la mejora del Servidor XMPP. Esto genero los resultados visualizados en la Figura 4 en donde se muestra el Diagrama de clases de Openfire, que permite visualizar las clases afectadas en el proyecto, incluyendo “IqRegisterHandler.class”, “Form.class”, “DataForm.class”, “DataForms.Item.class”, DbConnection.class y “regsettings”. Este último es responsable de la visualización de la consola web, que como su nombre indica, es el encargado de la configuración de los mecanismos de registro al interior del servidor Openfire. Para completar la explicación del funcionamiento de la implementación del XEP-0348 en Openfire, añadimos la Figura 3.

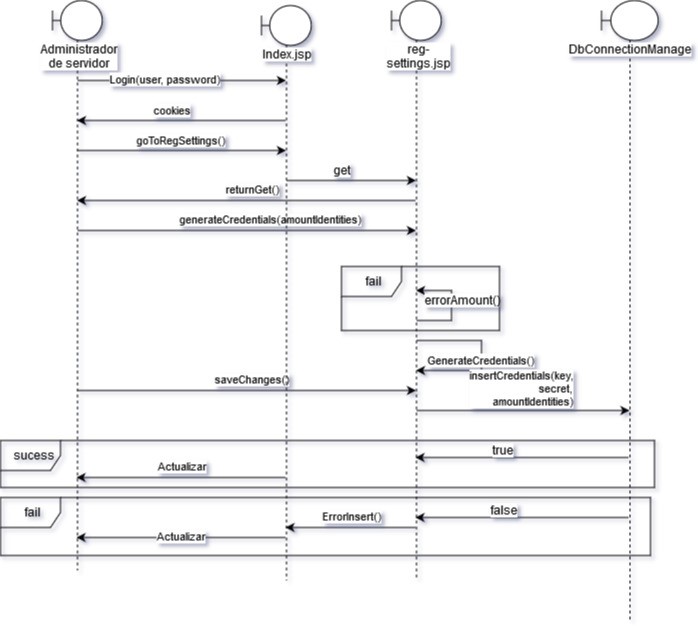
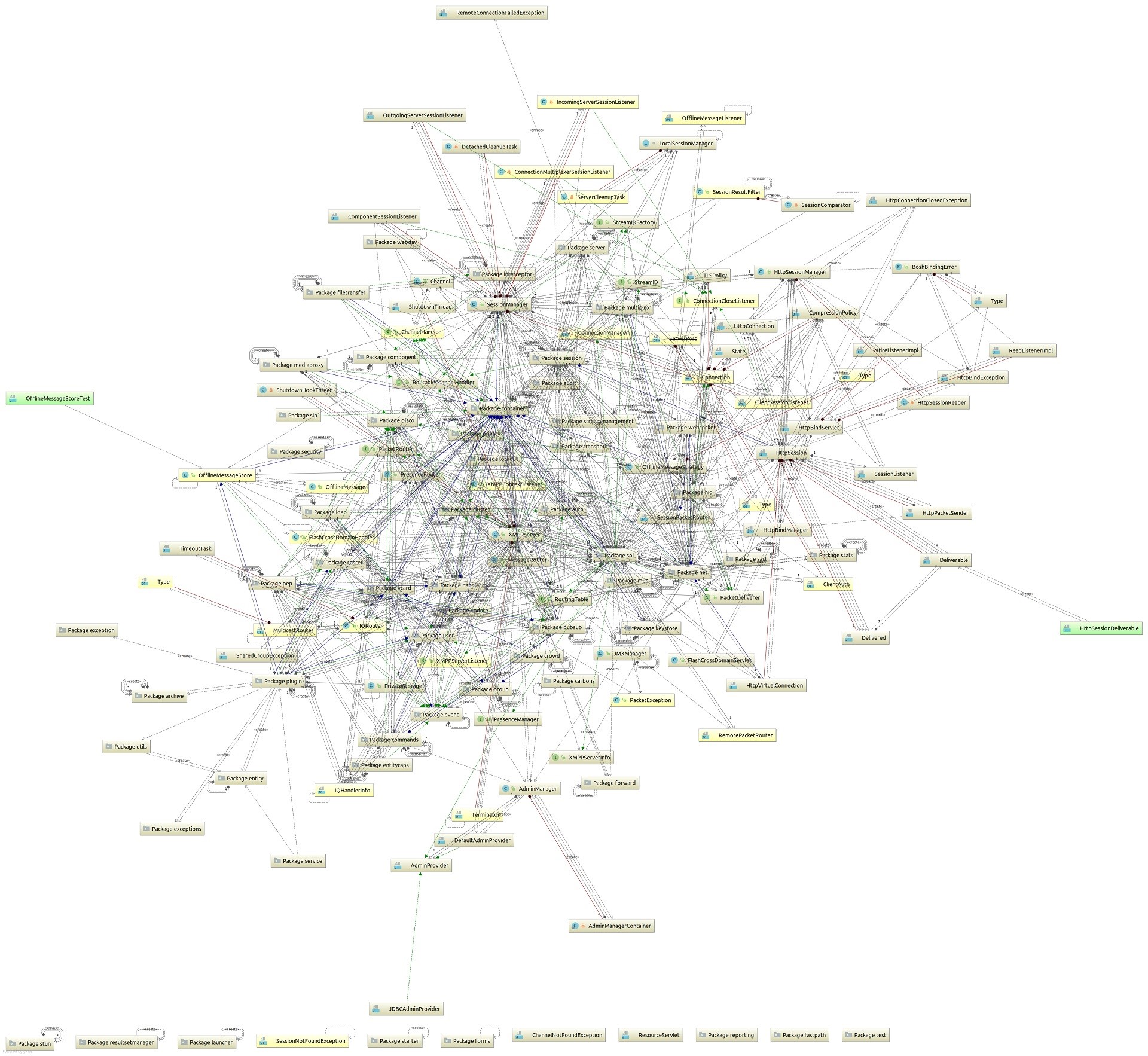


Figure 3: Openfire diagrama secuencia



Luego del desarrollo, pruebas y refactoring, se puede ver el resultado visible en la consola web del servidor, esta última se aprecia en la Figura 5. Aquí se ve el panel de “Configuración de Registro” de la consola web del servidor Openfire, donde la segunda sección que lleva el título de “Registration Signing Forms”, es una de las partes desarrolladas. En esta zona, el administrador de la red XMPP puede habilitar o deshabilitar el uso del mecanismo descrito en el XEP-0348: Signing Forms, como también generar nuevas credenciales de consumidor, donde cada una consiste en “llave de consumidor”, el “secreto del consumidor” y el número “creaciones autorizadas” este representa la cantidad de nuevas identidades que se pueden crear utilizando estas credenciales. Lo que significa, que cuando se llega a límite de la cantidad permitida, no se podrá continuar creando más identidades con dicha credencial. Por lo tanto, si es que el dueño de esta desea continuar creando identidades, tendrá que solicitar al administrador de la red XMPP-IoT que le genere nuevas credenciales.

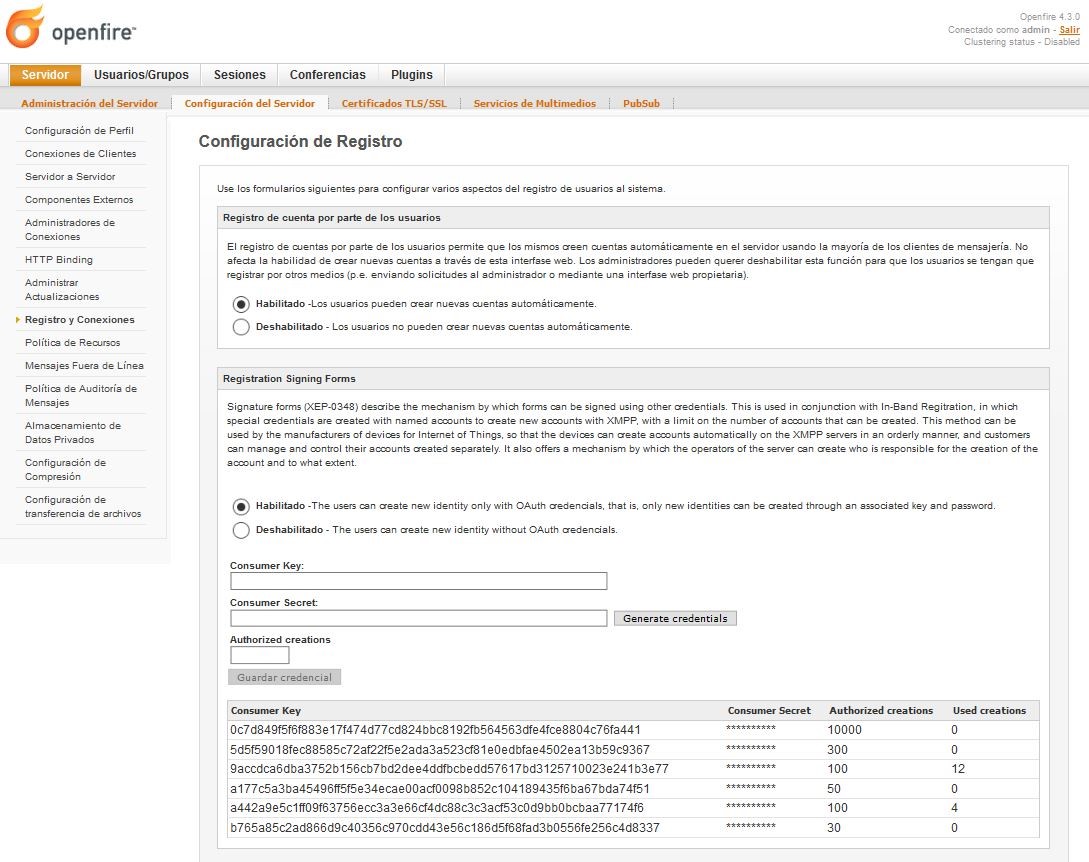


Figure 5: Openfire Consola Web

También se agregaron tablas a la base de datos al servidor Openfire, esto se puede apreciar en la Figura 6. Cabe mencionar que la manera de crear las relaciones es mediante indexs, y no por relaciones SQL propiamente dicho. La tabla agregada es “ofOAuth” para los fines del proyecto.

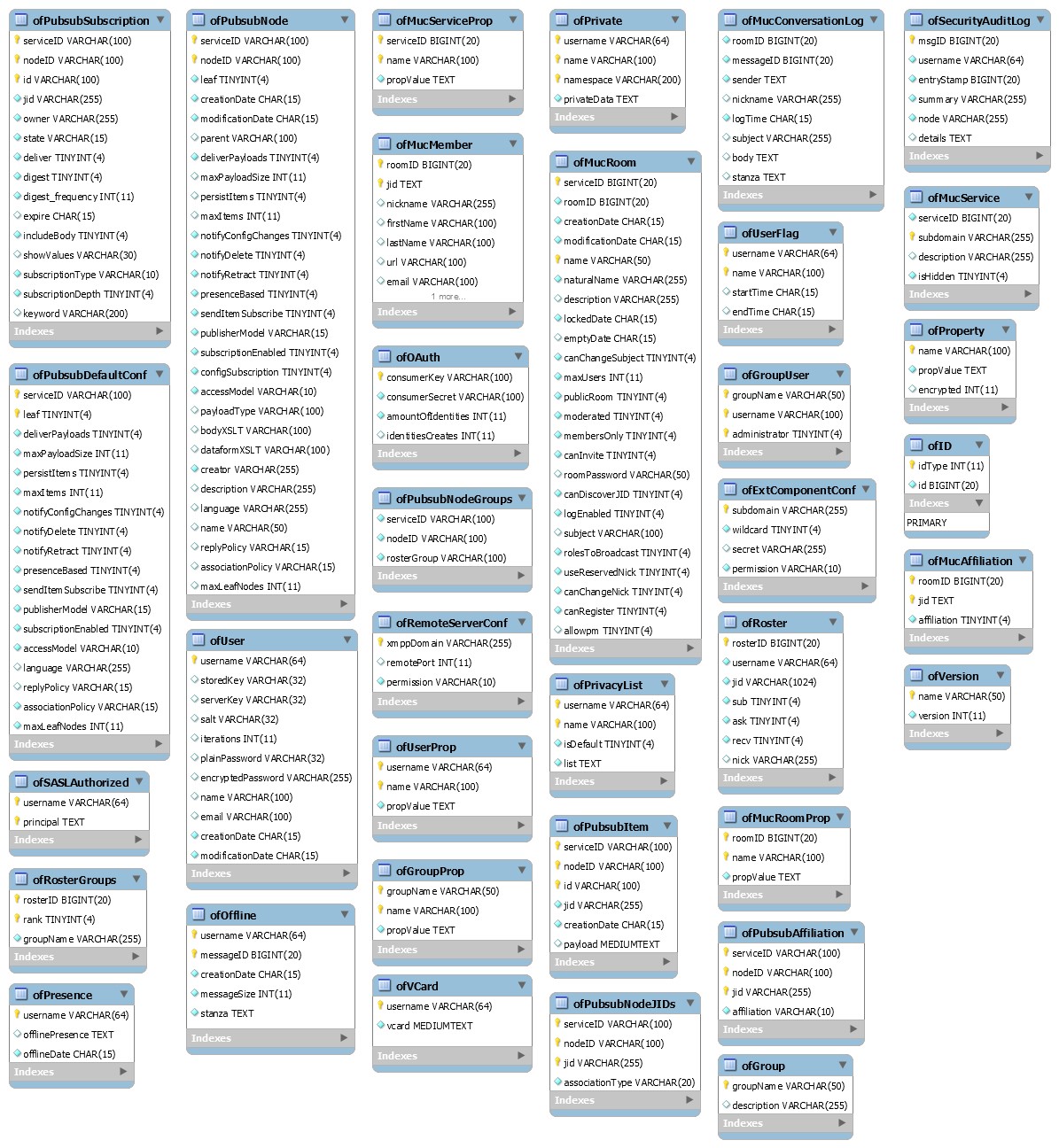


Figure 6: Diagrama de Base de datos

En el desarrollo de la mejora de Openfire, se escribieron 4595 líneas de código. La media de complejidad ciclomatica fue de 39. El valor es bastante alto, esto se debe a que la clase “IqRegHandler.class” posee un e(G) = 98, lo cual altera considerablemente la media en general. La clase inicialmente estaba puntada en 58, por ende, difícilmente se podía reducir la complejidad de esta añadiéndole funcionalidades.

El desarrollo de la Liberia SMACK, se inició identificando las entidades que posee el código fuente, el cual tiene en su interior varios subproyectos, que se pueden visualizar en la Figura 7, examinando estos, identificamos que en “smack-extensions” es donde están las clases involucradas en la creación de nuevas identidades en la red XMPP-IoT.

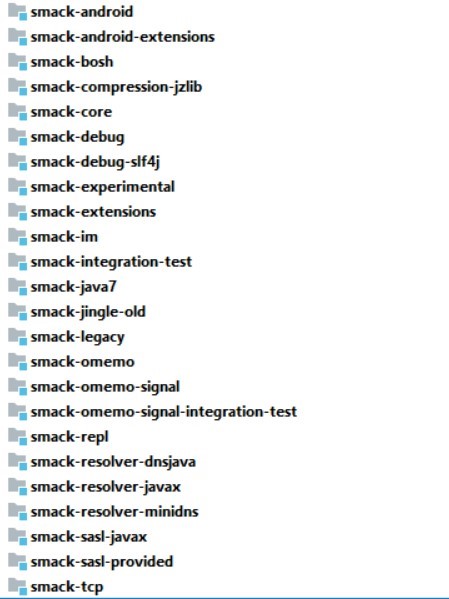


Figure 7: Proyectos anidados al interior de SMACK.

Luego de esto, se procedió a identificar las clases involucradas en el mecanismo descrito en el XEP-0077:

In-Band Registration. Agrupando estas, restan las clases mostradas en la Figura 8, luego de las modificaciones quedó acorde al diagrama ilustrado en la Figura 9.

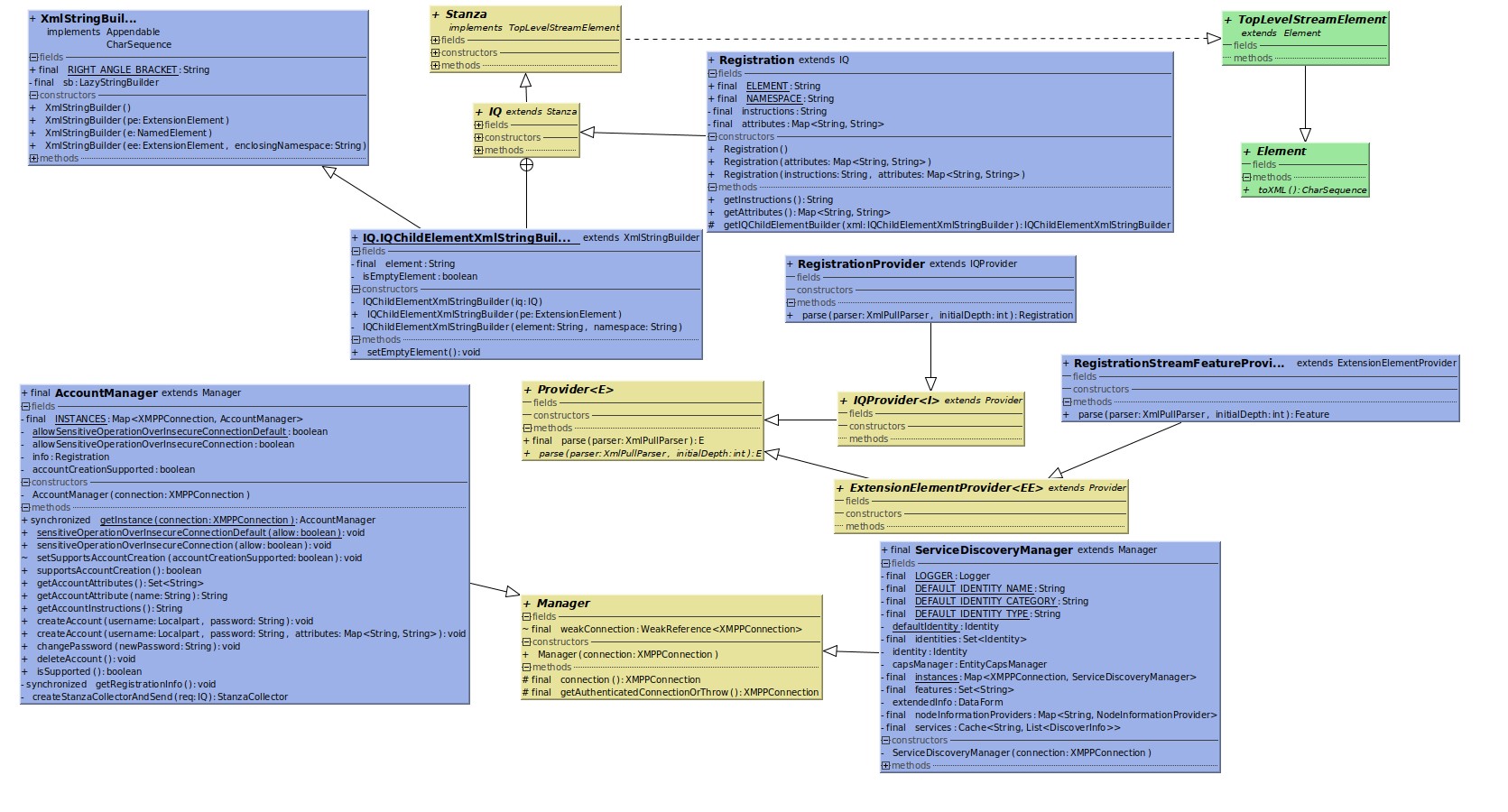


Figure 8: SMACK Diagrama de clases previo a la intervención de la librería.

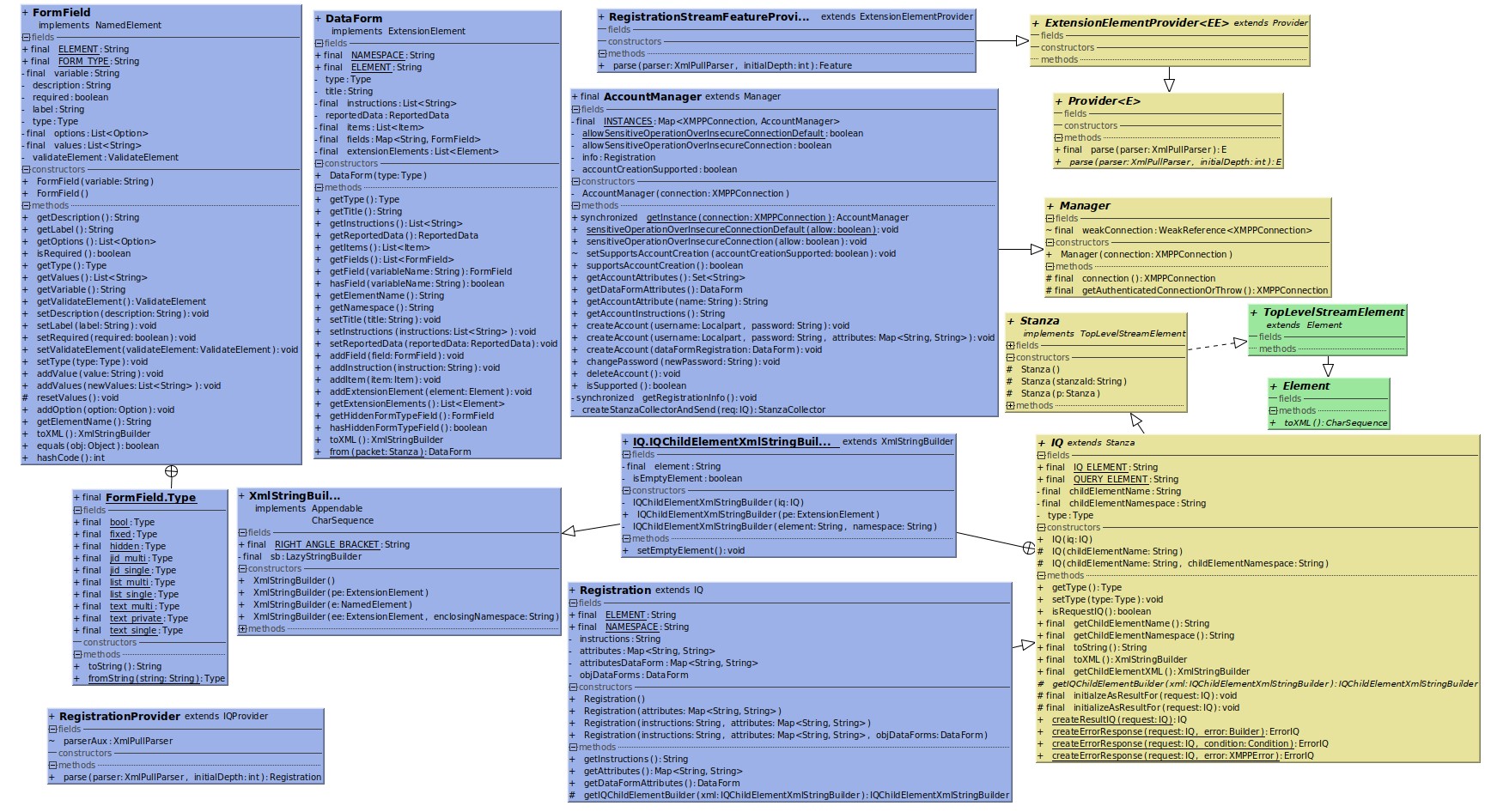


Figure 9: SMACK Diagrama de clases posterior a la intervención de la librería.

Finalmente, para depurar, en su conjunto, el sistema XMPP-IoT se creó un cliente usando la librería SMACK modificada, por ende, se incorporó incorporo las credenciales de consumidor. El cliente funciona como una Cosa que posee sensores de humedad, temperatura, gas, luz ambiente, movimiento y actuadores: interruptores de

luces y un display LED de 16x2, mediante el cual veremos información relevante de nuestro dispositivo. El programa establece la conexión con el servidor Openfire, luego envía la stanzas para crear automáticamente una identidad en este, utilizando el registro en-banda en conjunto del mecanismo de firmado de formularios. El servidor Openfire, el que debido a las añadiduras está facultado para responder la solicitud, responde a esta, para finalmente crear la identidad en la red XMPP-IoT. Los detalles de las stanzas enviados en esta conversación entre el cliente y el servidor, se pueden apreciar en las Figuras continuación.

Figure 10: Consulta de Features soportadas de cliente a servidor.



Figure 11: Respuesta de Features soportadas del servidor al cliente

En el momento que se realiza la conexión con el servidor, el cliente XMPP se debe asegurar que el servidor soporte el registro en-banda utilizando formularios firmados, para eso el cliente envía la stanza en la Figura 10 y el servidor debe responder las “Features” que soporte, la que anuncia el soporte del XEP-0348, es “urn:xmpp:xdata:signature:oauth1”. En la Figura 11 se aprecia la stanza en donde el servidor responde todas las “Features” que soporta.

El proceso de creación de identidades también se puede entender, por medio del siguiente diagrama de secuencia en la Figura 12.

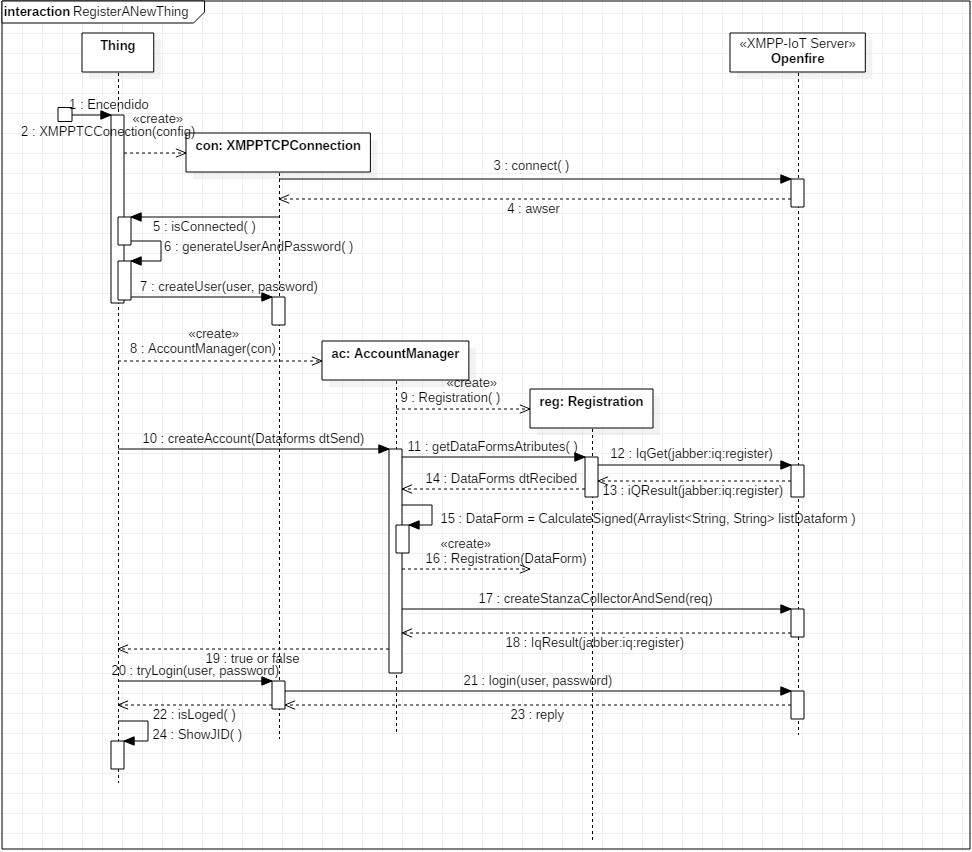


Figure 12: Registro de una nueva identidad usando SMACK modificada en un cliente Thing.

En siguiente diagrama de interacción, en la Figura 13, se expresa las entidades que interactúan cuando un usuario solicita los valores percibidos - por medio de mensajes de chat - a un sensor particular.

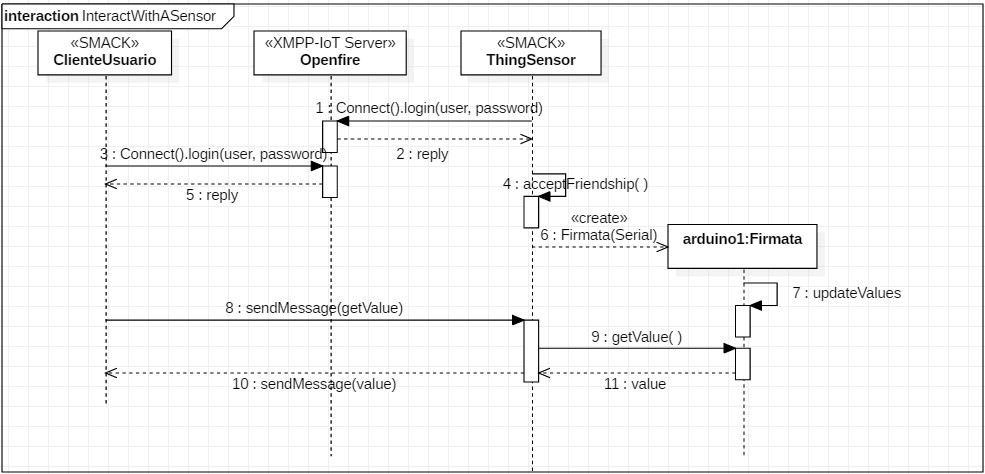


Figure 13: Solicitud del valor de un Sensor.

De forma similar a la lectura de un sensor, se pueden escribir valores en un actuador. El proceso para realizar esto, se encuentra en el diagrama de la Figura 14.

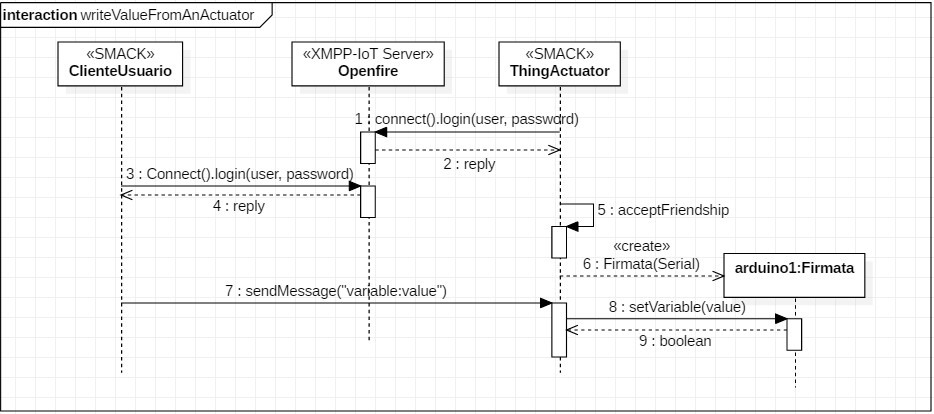


Figure 14: Escritura del valor en un actuador.

**Seguridad**

Con lo mencionado hasta el momento, se concluye el desarrollo de la mejora a la plataforma XMPP-IoT mediante la implementación de registro en-banda con firmado de formularios. Sin embargo, queda un tema pendiente, la seguridad en general. Si bien en esta publicación hablamos de seguridad por medio la delegación de responsabilidades para el registro de nuevas de identidades, usando el mecanismo con la XEP-0348: Signing Forms. De nada sirve que la red posea servidores con esta característica, si es existe la posibilidad que la identidad del servidor pueda ser fácilmente adulterada, es decir, que en el momento que un cliente XMPP se intenta conectar al servidor correspondiente, un oponente - hacker - puede realizar un ataque “Man in the Middle”, para suplantar la identidad de nuestro servidor, abriendo así la posibilidad de sustraer información sensible. Para minimizar el riesgo, una opción razonable es utilizar SSL/TLS, pero debido a los agujeros de seguridad en SSL producidos por la corta proyección al momento del diseño de este protocolo, lo deja vulnerable en todas sus versiones, desde la 1.0 a la 3.0. Debido a esta razón la mejor opción es usar TLS, y no cualquier versión, ya que también hay vulnerabilidades conocidas, las cuales, con un mediano poder de cómputo, pueden ser explotadas. La versión que utilizaremos será TLS 1.2 o la versión 1.1. TLS se puede dividir en 3 grandes fases o etapas, que consisten en:

1. Negociación de que algoritmo criptográfico se utilizará en la comunicación.
2. Intercambio de llaves públicas y la autentificación basada en certificados digitales.
3. Se cifra con criptografía simétrica el tráfico.

**Security Level**

Cuando nos referimos a algoritmos criptográficos, cabe mencionar que cada uno proporciona diversos “puntos fuertes” de seguridad, que están en función de qué tipo de algoritmo y qué tamaño de clave este utilice. Los Security Levels o Bits of Security son usados para estimar la fortaleza del cifrado para proteger la información sensible, en función de fortaleza potencial histórica del adversario, es decir, permite cuantificar qué tan fuerte debe ser un algoritmo para ser catalogado como “algoritmo criptográficamente fuerte”. Generalmente está basado en el “mejor” ataque conocido sobre el algoritmo. Esto implica que cada vez el ataque avanza, también lo debe hacer el algoritmo y/o su clave.

Para cada algoritmo criptográfico existen diversas publicaciones que proporcionan recomendaciones y fórmulas para aproximarse al tamaño de clave adecuado para cada algoritmo. Si bien, estos estudios están disponibles, aún sigue siendo una tarea sumamente engorrosa seleccionar un algoritmo y tamaño de clave adecuado, debido que, para esto, se debe leer y comprender todos estos documentos. Como una solución ante tal mar de información, nace keylength.com, una web que pretende simplificar tal labor dando un resumen con los tamaños de clave recomendados basados en los Security Level, en donde uno de los factores discriminantes es el Bits of Security (BOS). A continuación, se puede ver dos tablas comparativas de los algoritmos comúnmente usados con sus respectivos valores. La Tabla 8 y la Tabla 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Protección | Simétrica | Factoring Modulus | Logaritmo discreto  Llave Grupo | Curva Elíptica | Hash |
| Nivel estándar legado  No debería ser usado en sistemas nuevos | 80 | 1024 | 160 1024 | 160 | 160 |
| Protección a corto plazo  Seguridad durante al menos diez años (2018-2028) | 128 | 3072 | 256 3072 | 256 | 256 |
| Protección a largo plazo  Seguridad de treinta a cincuenta años (2018-2068) | 256 | 15360 | 512 15360 | 512 | 512 |

Table 7: Recomendaciones Criptográficas ECRYPT, año 2018.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fecha | Security | Algoritmos | Factoring | Log. Discreto | | Curva | Hash (A) | Hash (B) |
|  | Strength | Simétricos | Modulus | Llave | Grupo | Elíptica |  | Ej.: HMAC |
| (Legado) | 80 | 2TDEA | 1024 | 160 | 1024 | 160 | SHA-1 |  |
| 2016 - 2030 | 112 | 3TDEA | 2048 | 224 | 2048 | 224 | SHA-224  SHA-512/224  SHA3-224 |  |
| 2016 - 2030 | 128 | AES-128 | 3072 | 256 | 3072 | 256 | SHA-256 | SHA-1 |
| & después |  |  |  |  |  |  | SHA-512/256  SHA3-256 |  |
| 2016 - 2030 | 192 | AES-192 | 7680 | 384 | 7680 | 384 | SHA-384 | SHA-224 |
| & después |  |  |  |  |  |  | SHA3-384 | SHA-512/224 |
| 2016 - 2030 | 256 | AES-256 | 15360 | 512 | 15360 | 512 | SHA-512 | SHA-256 |
| & después |  |  |  |  |  |  | SHA3-512 | SHA-512/256  SHA-384  SHA-512  SHA3-512 |

Table 8: Recomendaciones Criptográficas NIST, año 2016.

Luego del análisis de los algoritmos y sus respectivas claves, considerando a un oponente con alto poder de cómputo, establecimos como protocolo criptográfico TLS Curva elíptica Diffie-Hellman RSA con AES256 SHA128-256 y como método de hashing HMAC SHA256 en la firma, en el mecanismo OAuth 1.0 modificado. En el primer caso en conjunto tiene un Security Strength de 256, al igual que OAuth, superando lo recomendado para la actualidad según el NIST [4] de EE. UU. y ECRYPT CSA de la Unión Europea [6] siendo ambas organizaciones posibles rivales, para tenerlos en consideración en caso de tener información sensible, según el criterio de ellos. En resumen, la plataforma XMPP-IoT está protegida por medio de los métodos criptográficos que podemos ver en las Tablas 9 y 10.

|  |  |
| --- | --- |
| Protocolos | |
| SSLv2 | No |
| SSLv3 | No |
| TLSv1 | No |
| TLSv1.1 | Sí |
| TLSv1.2 | Sí |

Table 9: Protocolos utilizados.

|  |  |  |  |
| --- | --- | --- | --- |
| Conjunto de cifrado | Bitsize | Confidencialidad directa | Información |
| ECDHE-RSA-AES256-GCM-SHA384 | 256 | Sí | Curva: sect571r1 |
| ECDHE-RSA-AES256-SHA384 | 256 | Sí | Curva: sect571r1 |
| ECDHE-RSA-AES256-SHA | 256 | Sí | Curva: sect571r1 |

Table 10: Métodos de cifrado utilizados.

De igual manera, se configuró las entradas DNS SRV[2], para que la comunicación entre servidores y clientes fuese fiable, mediante las siguientes entradas DNS:

* xmpp-client. tcp.domain.tld. TTL IN SRV priority weight port target
* xmpp-server. tcp.domain.tld. TTL IN SRV priority weight port target

Realizadas todas estas acotaciones en cuanto a seguridad, podemos decir que, en cuanto al estado del arte en el contexto criptográfico, el sistema se encuentra a la vanguardia.

# Validación

Tal como se ve en la Figura 1 el experimento consta de un servidor XMPP *Openfire* modificado con varios clientes XMPP *SMACK* modificados, estos últimos son:

* un Cliente XMPP Thing-Sensor (TS–1) que posee un sensor de humedad y temperatura
* un Cliente XMPP Thing-Sensor (TS–2) que posee un sensor de gas
* un Cliente XMPP Thing-Controlador (TC–1)

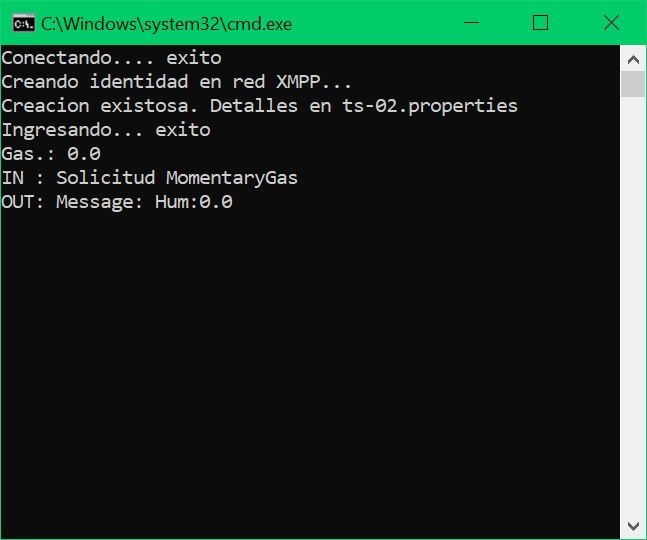
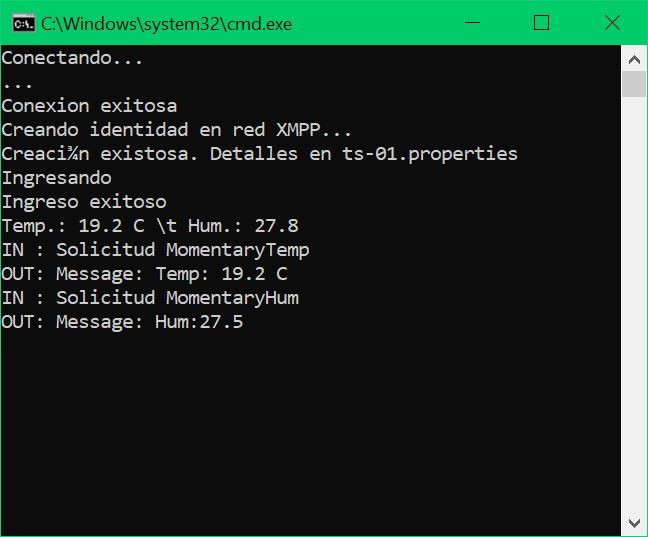
Ya realizada las modificaciones, el procedimiento es el siguiente:

1. Ingresar a la consola de administración web de *Openfire*.
2. Ingresar a la sección de registro.
3. Generar Credenciales de Consumidor, es decir, *ConsumerKey*, *Consumer Secret*.
4. Ingresar 3 como cantidad de creación de identidades permitida al *ConsumerKey*.

Para los clientes ingresamos el *ConsumerKey* y el *ConsumerSecret*, para que, de este modo, cuando se realice la conexión con el servidor, exista el intercambio de Stanzas con dataforms [5]. (está raro, no sé qué debería decir) este modo, se le delegue al cliente la facultad de crear identidades en el servidor XMPP Openfire. Así con el cliente TC–1 pudimos realizar consultas de (temperatura (?)) temperatura, humedad al cliente TS–1 y al gas a TS–2. Realizado esto, se creó y ejecuto el Bot malicioso en ambos servidores, tanto en el modificado como en el sin modificar. Los resultados se pueden apreciar en las siguientes Tablas:

Table 11: Test de penetración a servidor Openfire sin implementación

|  |  |  |
| --- | --- | --- |
| Tiempo (sec.) | Intentos de Creación por segundo | Identidades creadas exitosamente |
| 50 | 2 | 100 |
| 100 | 2 | 200 |
| 300 | 3 | 900 |



(a) Thing-Sensor 1 (b) Thing-Sensor 2

Figure 15: Retorno por consola de los clientes.

Table 12: Test de penetración a servidor Openfire implementado el XEP-0348

|  |  |  |
| --- | --- | --- |
| Tiempo (sec.) | Intentos de Creación por segundo | Identidades creadas exitosamente |
| 50 | 2 | 0 |
| 100 | 2 | 0 |
| 300 | 3 | 0 |