

INFORME DE PROYECTO DE GRADO PRESENTADO AL TRIBUNAL
EVALUADOR COMO REQUISITO DE GRADUACIÓN DE LA CARRERA
INGENIERÍA EN COMPUTACIÓN

INSTITUTO DE COMPUTACIÓN - FACULTAD DE INGENIERÍA

Contramedidas para la manipulación maliciosa de dispositivos en LoRaWAN

Estudiantes:

Sebastián PASSARO (4.588.411-6)

Martin PACHECO (4.638.953-7)

Supervisores:

Gustavo BETARTE

Juan Diego CAMPO

Montevideo, 17 de agosto de 2020



Resumen del trabajo

Las tecnologías que surgen del paradigma IoT (*Internet of Things*) popularmente aún se consideran emergentes y en varias ocasiones inseguras, sea por diseño o mala configuración frecuente. Este aspecto no es menor dado que los despliegues IoT están acaparando áreas esenciales involucradas a la calidad de vida, como el hogar, la salud, la ciudad y la industria. Particularmente, este trabajo investiga la seguridad de la tecnología LoRaWAN (*Long Range Wide Area Network*) contribuyendo a las aplicaciones para ciudades inteligentes. LoRaWAN fue diseñada para el escaso consumo energético y comunicación a larga distancia en su componente inalámbrica contemplando además controles de integridad y confidencialidad.

La base del trabajo es un modelo de amenazas para LoRaWAN construido en múltiples etapas. Su resultado final es la identificación de potenciales ataques y a alto nivel los pasos necesarios para llevarlos a cabo. Esta última característica permite definir contramedidas efectivas que reduzcan o eliminen el riesgo asociado.

Con las amenazas y ataques estudiados, se intenta demostrar empíricamente la viabilidad y efectividad de algunos de ellos. Especialmente se investigó el potencial de los ataques en radiofrecuencia (*jamming*) y su combinación con amenazas para lograr ataques sofisticados. Además se realizaron pruebas de ataques físicos de manera directa sobre el *hardware*, aplicando diversas metodologías sobre los componentes e interfaces. Finalmente, se ofrecen recomendaciones basadas en la literatura y en parte de la experimentación.

Palabras clave: Seguridad, LoRaWAN, LoRa, IoT, modelado de amenazas, *jamming*.

Índice

1. Introducción	8
2. Estado del arte/Revisión de antecedentes	10
2.1. Auge del paradigma IoT, aplicaciones y tecnologías	10
2.2. Seguridad IoT	12
2.3. LoRa y LoRaWAN	14
2.3.1. Introducción	14
2.3.2. Arquitectura	15
2.3.3. Seguridad	16
2.3.3.1. Esquema general	16
2.3.3.2. Contadores	17
2.3.3.3. Activación	17
2.3.3.4. Confidencialidad	20
2.3.3.5. Integridad	20
3. Modelado de amenazas	22
3.1. Caso de estudio y metodología	22
3.2. Diagramas de flujo de datos	22
3.3. Identificación de amenazas (STRIDE)	29
3.3.1. <i>Spoofing</i>	31
3.3.2. <i>Tampering</i>	31
3.3.3. <i>Repudiation</i>	31
3.3.4. <i>Information disclosure</i>	32
3.3.5. <i>Denial of service</i>	32
3.3.6. <i>Elevation of privilege</i>	32
3.4. Identificación de activos	32
3.5. Análisis de ataques	33
3.5.1. Objetivos de ataque	33
3.5.2. Árboles de ataque	35
3.5.2.1. A01, A02 - Modificación indebida de mensajes salientes y suplantación de dispositivo	35
3.5.2.2. A03 - Acceso administrativo al dispositivo	41
3.5.2.3. A04 - Alterar comportamiento de dispositivo	42
3.5.2.4. A05 - Modificación indebida de datos	43
3.5.2.5. A06 - Dejar fuera de la comunicación al dispositivo	44
3.5.3. Análisis con SHARPE	47
3.5.3.1. Diseño	48
3.5.3.2. Análisis probabilístico	50

4. Experimentación	55
4.1. Dispositivos utilizados	55
4.1.1. SparkX ESP32 LoRa 1-Channel Gateway	55
4.1.2. The Things Gateway	55
4.1.3. The Things Node	56
4.2. Captura de mensajes emitidos por el sonómetro	57
4.3. Herramientas de LoRaWAN Auditing Framework (LAF)	59
4.3.1. Persistencia de datos recibidos en The Things Gateway	59
4.4. <i>Jamming</i>	61
4.4.1. Equipamiento y librerías	61
4.4.2. <i>Jamming</i> continuo	61
4.4.3. <i>Jamming</i> reactivo	67
4.5. Modificación de mensajes	70
4.5.1. Construcción	70
4.5.2. Ejecución	73
5. Contramedidas	76
5.1. Auditoría	76
5.1.1. Análisis realizados por LAF	77
5.1.1.1. <i>Replay</i> de mensajes	77
5.1.1.2. Mensajes falsos y denegación de servicio (sesiones simultáneas)	77
5.1.1.3. Dispositivos ABP	77
5.1.1.4. Claves conocidas o no suficientemente aleatorias	78
5.1.1.5. Otros controles	78
5.1.2. Prueba de uso de LAF	78
5.1.3. Conclusiones sobre LAF	80
5.2. Recomendaciones	80
5.2.1. <i>Bit-flipping</i> en enlace ED-GW	80
5.2.2. <i>Jamming</i>	81
5.2.3. <i>Acceso físico</i>	82
5.2.4. <i>Bypass</i> de mecanismos de protección en <i>hardware</i>	82
5.2.5. OTA y circuito integrado de LoRa	83
5.2.6. Derivar claves de sesión de AppKey	84
5.2.7. Robo de claves en dispositivo	84
5.2.8. Robo de claves en <i>network server</i>	85
5.2.9. Acceso administrativo al dispositivo	85
5.2.10. Alterar comportamiento del dispositivo	86
5.2.11. <i>Replay</i> de mensajes antiguos en la misma sesión	86
5.2.12. <i>Bit-flipping</i> en enlace NS-AS	86
5.2.13. Exceso diario de mensajes	87
5.2.14. <i>Replay</i> en <i>Join Procedure</i>	87
6. Conclusiones y trabajo futuro	89
A. Actores relevantes en modelado de amenazas	101

B. Modificaciones a The Things Node	102
C. Modificaciones a util-lorawan-packets	104
D. Estructura de carpetas de la entrega	106

Índice de figuras

1.	Algunas configuraciones en tecnologías IoT <i>short to medium range</i> y <i>wide area</i> . . .	11
2.	Componentes de un dispositivo IoT.	12
3.	Arquitectura general de LoRaWAN.	15
4.	Esquema de seguridad en LoRaWAN 1.0.3.	17
5.	<i>Join Procedure</i> en LoRaWAN 1.0.3.	18
6.	Desglose de PHYPayload hasta FHDR.	20
7.	DFD general de LoRaWAN 1.0.3 hasta el envío de datos a una integración. . . .	23
8.	DFD de dispositivo IoT que reporta medidas del ambiente.	25
9.	DFD de <i>gateway</i> LoRaWAN.	27
10.	DFD de <i>network server</i> en LoRaWAN 1.0.3.	28
11.	DFD de <i>application server</i> en LoRaWAN 1.0.3.	29
12.	Árbol de ataque para el objetivo “Modificación indebida de mensajes salientes”, incluyendo también a “Suplantación de dispositivo”.	36
13.	Etapas de <i>jamming</i> selectivo comparado a <i>triggered</i>	37
14.	Clasificación alternativa de <i>jamming</i> s.	38
15.	Árbol de ataque para el subobjetivo “Obtener claves de sesión”.	40
16.	Árbol de ataque para el objetivo “Acceso administrativo al dispositivo”.	41
17.	Árbol de ataque para el objetivo “Alterar comportamiento de dispositivo”. . . .	42
18.	Árbol de ataque para el objetivo “Modificación indebida de datos”.	43
19.	Árbol de ataque para el objetivo “Dejar fuera de la comunicación a dispositivo”. .	45
20.	Árbol en SHARPE de modificación indebida de mensajes salientes junto a sus subárboles correspondientes.	49
20.a.	Modificación indebida de mensajes salientes	49
20.b.	Evitar envío/recepción de mensajes	49
20.c.	Suplantación de dispositivo	49
20.d.	Carga de <i>firmware</i> corrupto	49
20.e.	Acceso remoto administrativo al dispositivo	49
20.f.	Obtener claves de sesión	49
20.g.	Robar claves de dispositivo	49
21.	SparkX ESP32 LoRa 1-Channel Gateway sobre <i>protoboard</i> con antena conectada. .	55
22.	The Things Gateway con antena conectada.	56
23.	The Things Gateway con antena conectada y sin tapa superior.	56
24.	The Things Node, vista frontal.	57
25.	The Things Node, vista trasera sin tapa.	57
26.	Consola del <i>gateway</i> de The Things Network.	59
27.	Mensajes de TTN en base de datos.	60
28.	<i>Jamming</i> continuo sobre TTNnode en la frecuencia 916.8 en dos intervalos.	64
29.	<i>Symbols</i> no ortogonales con diferentes SF y BW, misma pendiente.	65
30.	<i>Symbols</i> ortogonales con diferentes SF y BW, diferente pendiente.	66
31.	Ortogonalidad de combinaciones entre SF y BW, las “X” indican no ortogonalidad. .	66
32.	Escenario para prueba completa de modificación de mensajes.	74
33.	Flujo capturado de mensajes en TTGW con el escenario planteado.	75

34.	Alerta LAF-006 - Detección de un dispositivo ABP.	79
35.	Alerta LAF-007 - Contador recibido menor que el esperado.	79
36.	<i>Dongle</i> LoRa USB de iFrogLab.	92

Índice de tablas

1.	Interfaces de los DFD con sus identificadores, extremos y qué fluye a través de estas.	24
2.	Listado de amenazas siguiendo el modelo STRIDE.	30
3.	Objetivos de ataque para los dos activos críticos por orden de prioridad.	34
4.	Asignación de probabilidades en SHARPE para el caso particular planteado. . .	51
5.	Variaciones de probabilidad para hoja de acceso físico al dispositivo y sus impactos en la probabilidad de falla total.	54
6.	Actores relevantes en las potenciales amenazas.	101

1. Introducción

En la última década se ha dado un enorme incremento en la cantidad de objetos (como electrodomésticos, equipamiento industrial, dispositivos médicos) a los que se le agrega cierto poder de cómputo y son conectados a Internet. A esta realidad se la ha denominado Internet de la Cosas (*Internet of Things*, IoT de aquí en más). Estas tecnologías están en pleno auge a nivel mundial y ya están empezando a ser implantadas en Uruguay. Ejemplos de ello son la red eléctrica inteligente, o los avances hacia la ciudad inteligente.

Diversos estudios recientes [1, 2, 3] han revelado un sinnúmero de vulnerabilidades que padecen estos dispositivos y los protocolos que utilizan, que permiten que un atacante los comprometa y utilice con fines maliciosos. En especial, la cantidad de dispositivos conectados y su relativa baja seguridad los ha hecho blanco fácil de operadores de *botnets*, como el caso de Mirai [2] que llegó a contar con más de 600 mil dispositivos infectados.

Desde el inicio de este trabajo, el área de ciudades inteligentes de la Intendencia de Montevideo (IM) mostró interés en colaborar con este proyecto, dado que en su trabajo hay proyectos IoT en marcha y planes piloto. Con esto en mente, se buscó un acercamiento donde se generara valor por ambas partes en el marco del proyecto de grado. Con este enfoque, la línea de trabajo se concentró en ciudades inteligentes y particularmente con la tecnología LoRaWAN [4].

Considerando lo antes mencionado sobre la seguridad IoT y que LoRaWAN es una tecnología emergente, los objetivos del proyecto son contribuir al estudio de la seguridad en ciudades inteligentes y dispositivos IoT y específicamente analizar el nivel de riesgo que su despliegue puede tener en Uruguay. Para esto, en el proyecto se desarrolla en las siguientes etapas:

- Estado del arte de la seguridad en redes de dispositivos inteligentes.
- Análisis de seguridad de una realidad de despliegue genérico de dispositivos IoT y ciudades inteligentes LoRaWAN. Este se centrará en los dispositivos y la comunicación mediante la red inalámbrica LoRa. Con el fin de acotar este escenario tan amplio, se decidió estudiar principalmente dispositivos del tipo sensores que envían las medidas tomadas, caso de estudio planteado por IM.
- En el desarrollo del punto anterior se realiza un modelado de amenazas que incluye:
 - Análisis de flujos de datos en LoRaWAN.
 - Listado de amenazas siguiendo el modelo STRIDE [5].
 - Identificación de activos.
 - Análisis de procesos en los que se pueden ver expuestos los activos.
 - Identificación de potenciales actores involucrados.
 - Identificación de vectores de ataque y realización de pruebas de concepto que los validen y exploten (limitándose a las pruebas técnicamente viables y que se tienen los recursos necesarios disponibles).
 - Estudio de contramedidas para prevenir y/o mitigar las vulnerabilidades identificadas.

Como resultado, se han identificado diversos tipos de ataque, algunos de ellos inevitables por su naturaleza. Sin embargo, se concluye que siguiendo una serie de recomendaciones que incluyen monitoreo constante, es posible desplegar una red IoT LoRaWAN con un bajo nivel de riesgo.

2. Estado del arte/Revisión de antecedentes

2.1. Auge del paradigma IoT, aplicaciones y tecnologías

El paradigma IoT surgió de forma disruptiva en una época de avance tecnológico que ha permitido su inclusión en diversos ámbitos y aplicaciones. Las tecnologías IoT se han introducido en la industria, ciudades y hogares. Dentro de lo que parecería tener más incidencia en la población, se dan ejemplos de uso [6] en los llamados *smart homes* u hogares inteligentes (aplicado a seguridad, eficiencia energética y confort, salud y bienestar, ayuda para ancianos o discapacitados) y a más alto nivel en las llamadas *smart cities* o ciudades inteligentes (aplicado a optimización de servicios públicos tradicionales, monitorización de consumo de energía, calidad de aire y sonido, gestión de tráfico, reducción de costos operacionales, iluminación, estacionamiento, entre otros).

El despliegue masivo de dispositivos es inminente. En 2017, Gartner predijo que para este año (2020) los dispositivos conectados mundialmente alcanzarían los 20,4 mil millones y que el gasto total en *hardware* se aproximaría a los 3 billones de dólares [7]. Despliegues de tal magnitud tienen sus beneficios, así como sus desafíos. Un auge tan veloz y competitivo dio lugar a numerosas implementaciones de tecnologías y protocolos al punto de darse la incapacidad de definir estándares para estos desarrollos. Organizaciones como FIWARE [8] ofrecen plataformas que posibilitan trabajar en el contexto IoT a través de un conjunto de *middleware* y servicios que apuntan a integrar dispositivos, protocolos e información. Dentro de las alternativas se encuentran [8] AWS IoT, Kaa, IBM Watson IoT y Azure IoT Hub, entre otras. Empresas y otros organismos tienden a utilizar estas plataformas dado que tienen sus propias especificaciones y entre más se utilice una plataforma, más se aproxima a ser un estándar.

Por supuesto, el nacimiento de IoT no se dio instantáneamente. Un trabajo de investigación en la Universidad Metropolitana de Manchester [9] muestra una interesante y detallada línea temporal con la evolución de las TICs (Tecnologías de la Información y Comunicación) hasta llegar a IoT, que parte desde la invención del telégrafo electromagnético (1830) hasta el anuncio de Microsoft en 2018 de una inversión en IoT de 5 mil millones de dólares para los siguientes cuatro años. En este mismo trabajo se mencionan las tecnologías inalámbricas que hoy en día se utilizan en el ámbito IoT, comparando características como aplicaciones, topología, rango de señal, entre otras. Las clasifican en *short to medium range* y *wide area* y las mencionadas son:

- *Short to medium range*: Bluetooth, Bluetooth Low Energy, Ant, Wi-Fi, NFC, Zigbee, Z-wave.
- *Wide area*: Cellular 3G/4G, SigFox, LoRa, NB-IoT, LTE-M.

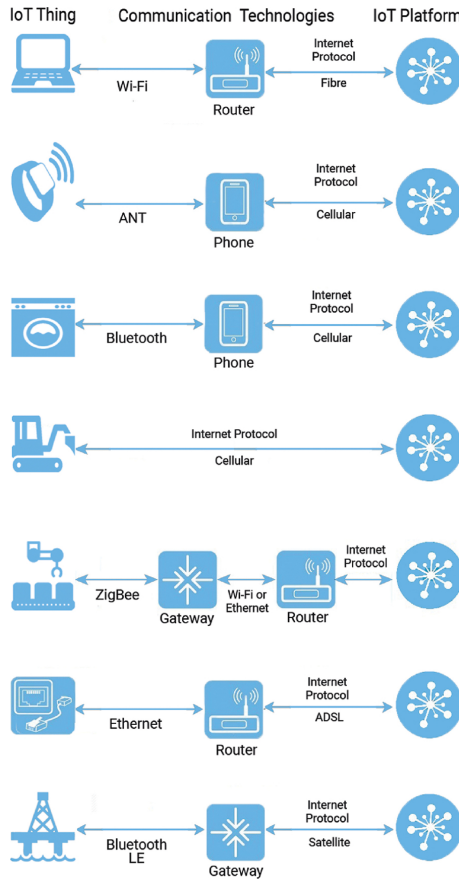


Figura 1: Algunas configuraciones en tecnologías IoT *short to medium range* y *wide area* [9].

La Figura 1 muestra lo variadas que son las configuraciones de las tecnologías, particularmente para las *short to medium range*. En el listado anterior se aprecia la “reutilización” de tecnologías existentes hace años como NFC, Wi-Fi, Bluetooth o 3G/4G, y la aparición de otras como LoRa, SigFox o Zigbee. Es interesante destacar que incluso se dio la evolución de Bluetooth para optimizar la gestión energética, naciendo así Bluetooth *Low Energy* (BLE). En esta línea de optimización es que se crearon las tecnologías LPWAN (*Low-Power Wide Area Network*), amoldadas a la aplicación en el mundo IoT pues en diversos casos no es viable la alimentación energética continua por cable y se vuelve necesario incluir baterías. De hecho, dentro de un dispositivo IoT, uno de sus componentes principales es el suministro energético (*power supply*) [9] como se ve en la Figura 2. También se requiere una unidad de sensor y/o actuador que sea capaz de detectar cambios de algún tipo, una unidad de comunicación para enviar mensajes a otras entidades, una unidad de memoria adicional para el procesamiento de datos y como nexo de todas ellas la unidad de procesamiento, que involucra un procesador (en general un microcontrolador), una memoria volátil y una memoria no volátil (donde usualmente se almacenan el *firmware* y configuraciones). A partir de estos componentes básicos el dispositivo puede añadir más componentes aumentando su complejidad.

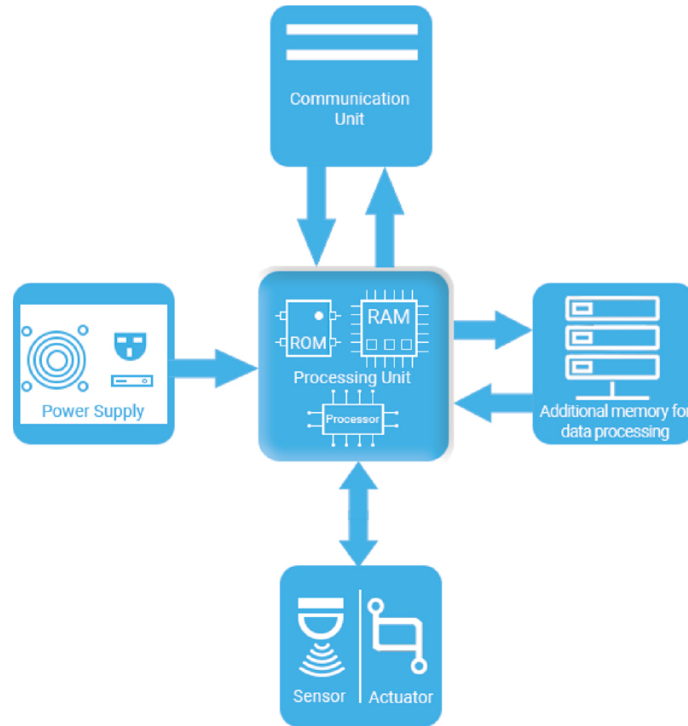


Figura 2: Componentes de un dispositivo IoT [9].

Adicionalmente, se han desarrollado protocolos a nivel de datos que contemplan las necesidades para la realidad IoT, buscando principalmente ser ligeros. Dentro de los destacados se encuentran HTTP/REST (sin cambios para IoT), *Message Queuing Telemetry Transport* (MQTT), *MQTT for Sensor Networks* (MQTT-SN), *Constrained Application Protocol* (CoAP, desarrollado por la *Internet Engineering Task Force*, IETF), *Advanced Message Queuing Protocol* (AMQP) y *Extensible Messaging and Presence Protocol* (XMPP) [6]. En general los protocolos de datos no contemplan inherentemente la seguridad, por lo que lo común es integrar protocolos de cifrado a nivel de transporte de los datos como *Transport Layer Security* (TLS) o *Datagram Transport Layer Security* (DTLS). Que el cifrado no sea intrínseco al protocolo provoca que en ocasiones sea olvidado de integrar y puede que tampoco se integre a propósito por problemas de performance o simplicidad.

2.2. Seguridad IoT

A la falta de estándares la acompaña la falta de medidas de seguridad. La gran mayoría de los dispositivos se caracterizan por su simpleza, por lo que sus mecanismos de protección no suelen ser los mejores ni se promueve por parte de los fabricantes el aseguramiento de los dispositivos previo al despliegue. Los problemas comunes en IoT son la exposición a Internet de los dispositivos y la configuración de fábrica en estos (lo que implica el uso de credenciales por defecto o su total ausencia), permitiendo ser localizados en escaneos masivos por motores de búsqueda como Shodan [10]. De lograr controlar una cantidad suficiente de dispositivos, un atacante puede dejarlos fuera de funcionamiento o utilizarlos para otros fines, como realizar denegación

de servicio distribuido (DDoS) a terceros, minado de Bitcoins, robo y manipulación de datos, chantaje/extorsión, acoso o robos [6]. Una investigación ha llegado a estudiar casos en donde es posible afectar la red eléctrica de un país abusando de una *botnet* de dispositivos IoT de alto wataje (aires acondicionados, hornos, termotanques, estufas) [1]. El caso concreto de mayor repercusión en los últimos años fue el de la *botnet* Mirai, que contó con un pico de 600 mil dispositivos infectados con el objetivo de llevar a cabo ataques DDoS que han superado los 600 Gbps [2].

Las tecnologías mencionadas en 2.1 tienen sus propios diseños de mecanismos de seguridad, diseños que en mayor o menor medida, son de confiabilidad discutible y con suficiente investigación pueden ser explotados. Existen investigaciones que revelan vulnerabilidades a nivel de protocolo para BLE [3, 11] donde en ocasiones las mitigaciones no son viables por cuestiones económicas y prácticas. Por ejemplo, es recomendado que los dispositivos posean mecanismos de verificación de códigos autogenerados al momento de aparearse, incluyendo *display* y teclado numérico, pero para un fabricante esto puede ser costoso e incluso violar alguna característica de diseño inicial del dispositivo como tamaño reducido. Más adelante en este documento, se comentará específicamente el caso de LoRa.

En otros casos, las vulnerabilidades se dan no solo en el protocolo, sino también en el propio dispositivo. La familia de chips ESP32 [12] (creada por Espressif Systems) es una opción popular y relativamente económica para desarrollos IoT, en parte porque fue diseñada para contar con consumo energético sumamente bajo, posee soporte integrado para Wi-Fi y Bluetooth y es posible integrarla en circuitos para soportar otras tecnologías (como LoRa en [13]). En 2019 el investigador de seguridad de *hardware* cuyo pseudónimo es LimitedResults, reportó que era posible extraer claves involucradas en los mecanismos de seguridad del *hardware* e incluso grabar en el dispositivo un nuevo *firmware* modificando salteando estos mecanismos [14, 15]. Consecuentemente, se publicaron los correspondientes CVE (CVE-2019-15894 y CVE-2019-17391), Espressif actualizó sus *frameworks* de desarrollo y liberó nuevas versiones de ESP32 con las correcciones necesarias a nivel de *hardware*. Otro ejemplo se dio con el investigador Matheus Garbelini, quien encontró vulnerabilidades en Wi-Fi para ESP32 y ESP8266 (una familia de chips previa) y publicó pruebas de concepto para los ataques identificados [16]. Lo interesante de este último caso es que uno de los ataques para ESP8266 estuvo varios meses sin solución en una liberación oficial, una situación que da alerta siendo que aunque se esté totalmente actualizado pueden existir vulnerabilidades *zero-day* sin ser mitigadas por tiempo desconocido.

El proyecto OWASP (*Open Web Application Security Project*) tiene dentro de sus líneas de colaboración e investigación para la comunidad, un proyecto particular para IoT [17] bajo la siguiente descripción: “El proyecto *OWASP Internet of Things* está diseñado para ayudar a fabricantes, desarrolladores, y consumidores a entender mejor los problemas de seguridad asociados al Internet de las Cosas, y permitir a los usuarios en cualquier contexto a tomar mejores decisiones de seguridad cuando construyan, desplieguen o evalúen tecnologías IoT.”. En él se incluyen guías varias y un top 10 de vulnerabilidades en el contexto IoT.

2.3. LoRa y LoRaWAN

2.3.1. Introducción

LoRa (*Long Range*) o LoRa RF [4] es la capa física o modulación inalámbrica utilizada para crear enlaces de comunicación de largo alcance. LoRa se basa en la técnica de modulación *Chirp Spread Spectrum* (CSS), la cual se caracteriza por su baja potencia y largo alcance. CSS ha tenido usos militares y de comunicación espacial por décadas gracias a las distancias alcanzadas y robustez a la interferencia, pero LoRa es la primera implementación de bajo costo para uso comercial. Es una tecnología LPWAN, por tanto, ofrece duración de varios años de batería (hasta 10 años) y está diseñada para sensores y aplicaciones que necesitan enviar pequeñas cantidades de datos sobre largas distancias unas pocas veces por hora en ambientes sumamente variantes.

Las frecuencias de LoRa varían por región en bandas ISM (*Industrial, Scientific and Medical*, reservadas internacionalmente para esos propósitos). Por ejemplo, en Europa se permiten las bandas de 863-870 MHz, en Estados Unidos (y el resto de América) se utiliza 902-928 MHz, en Australia 915-928 Mhz y análogamente con otras regiones. En cada región existen políticas diferentes para regular el tráfico. Se introduce el concepto de *duty cycle* que es el porcentaje de tiempo en el que un dispositivo ocupa el canal. En la banda EU863 se definió como máximo *duty cycle* un 1%, resultando en transmisión máxima de 36 segundos por hora en cada sub-banda para cada dispositivo [18].

Por otro lado, **LoRaWAN** (*LoRa Wide Area Network*) define el protocolo de comunicación y arquitectura para la red, mientras LoRa habilita el enlace de largo alcance. El protocolo y arquitectura de red tienen la mayor influencia en determinar la capacidad de la red, la calidad del servicio, la seguridad, la variedad de aplicaciones servidas por la red y la vida de la batería de un nodo. Contribuyendo a este último punto, evitar la sincronización con la red en períodos donde el dispositivo “despierta” es un punto clave en la reducción de consumo energético. Los dispositivos LoRaWAN son asíncronos y transmiten cuando disponen de datos listos enviar, ya sea en función de un evento o planificado, es decir siguen el método de ALOHA. La naturaleza de la familia de protocolos ALOHA [19] es acceder aleatoriamente al medio compartido, sin sincronización previa. Si detectan colisión en la transmisión, retransmiten con una demora aleatoria hasta que el mensaje sea enviado sin interrupciones. No existe una única versión de ALOHA, para más detalles consultar [19].

Los dispositivos en LoRaWAN pueden ser de tres clases [4]:

- **Clase A:** Permiten la comunicación bidireccional, enviando cuando lo necesitan, aunque recibiendo solo en dos cortas ventanas de tiempo luego de haber enviado. El instante de envío depende de la necesidad del dispositivo y pequeñas variaciones de tiempo aleatorias debido a que es un protocolo del tipo ALOHA. La clase A **debe** ser implementada por todos los dispositivos. Es la más eficiente energéticamente, pero la que deriva en mayor latencia de bajada (*downlink*).
- **Clase B:** Cumplen con las características de la clase A, pero a su vez soportan la configuración de ventanas de recepción extra. Esto se produce luego de la recepción de *beacons*

de sincronización para tener seguridad de cuándo el dispositivo está escuchando.

- **Clase C:** Su ventana de recepción es continua, cortándose únicamente cuando se debe transmitir.

2.3.2. Arquitectura

Previo a explicar los detalles de la arquitectura de LoRaWAN, se aclara que las versiones de LoRaWAN son definidas y actualizadas por LoRa Alliance, que comenzaron en la versión 1.0 dando lugar a la familia de versiones 1.0.* (hasta 1.0.3) y luego surgió la versión 1.1 contemplando diversas mejoras, entre ellas de seguridad, aunque la 1.0.3 es posterior en el tiempo a la 1.1. Visto esto, la aclaración es necesaria pues en 1.1 se incorporan más componentes a la arquitectura, pero no se analizará en el marco de este proyecto. Se decidió proceder con la versión 1.0.3 por ser la más actual y propensa a utilizarse en la industria local, además 1.0.* es la propuesta a investigar en el caso de estudio como se comenta en la sección 3.1.

A diferencia de otras redes IoT basadas en arquitectura *mesh* (nodos reenvían mensajes de otros nodos, logrando mayor cobertura), LoRaWAN sigue una topología de estrella de estrellas.

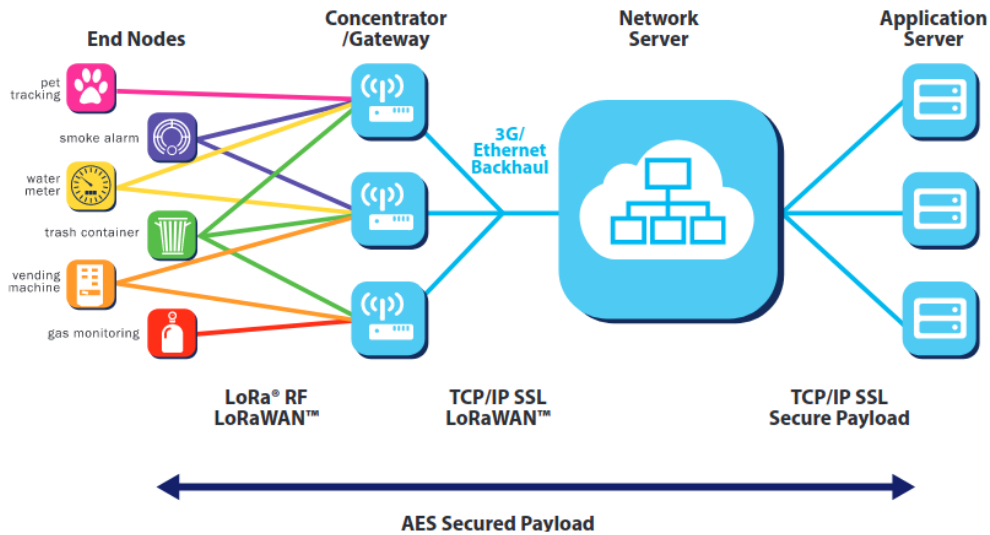


Figura 3: Arquitectura general de LoRaWAN [4].

La Figura 3 resume la arquitectura de LoRaWAN 1.0.*. Los componentes de esta red son (de aquí en más serán referidos por sus abreviaciones por simplicidad de redacción):

- **End-Nodes o End-Devices (ED):** Los dispositivos IoT unidos a la red LoRaWAN. Se comunican a través de LoRa al medio de forma bidireccional con los *gateway*.
- **Concentrador o Gateway (GW):** Dispositivos con antena para la recepción de mensajes LoRa provenientes de los EDs. Su única tarea es reenviar mensajes LoRa íntegros al *Network Server* por un enlace directo TCP/IP basado en tecnologías tradicionales como Ethernet o 3G. A su vez está encargado de transmitir mensajes a los EDs provenientes del *Network*

Server. Dependiendo de la necesidad pueden coexistir múltiples GWs, de hecho, un ED no se comunica con un GW particular, sino que todos los GWs captan los mensajes y los envían al *Network Server*, quien se encarga de descartar duplicados.

- **Network Server (NS)**: Servidor encargado de gestionar las sesiones LoRaWAN con todos los EDs (mantener parámetros de configuración y claves de sesión), validar los mensajes provenientes de los EDs y reenviarlos al *Application Server*. A su vez, este último envía al NS las configuraciones referidas a aplicaciones, EDs y GWs según como se defina la red. Más de una instancia de NS podría integrarse a LoRaWAN. En resumen, es un enrutador de los mensajes LoRaWAN y controla la autenticación y autorización de los EDs.
- **Application Server (AS)**: Es el destino de los datos provenientes de EDs como *payload* de LoRaWAN. Tiende a implementarse por usuarios finales de acuerdo con el propósito de los usuarios consumidores.

Más detalles sobre los elementos de la red y la seguridad se explicarán más adelante.

2.3.3. Seguridad

2.3.3.1 Esquema general

El aspecto que faltó mencionar de la Figura 3 es la seguridad. La Figura 4 brinda más detalles de este aspecto. La seguridad de los datos en LoRaWAN se contempla en el diseño. La comunicación se da obligatoriamente partiendo de la base de que existen dos claves de sesión [20] llamadas *Network Session Key* (**NwkSKey**) y *Application Session Key* (**AppSKey**), ambas almacenadas en el ED. La NwkSKey es la clave con la que se firman los mensajes LoRa, cumpliendo con el control de integridad. Cuando un mensaje es recibido por el NS, este valida por última vez la integridad y en caso positivo se extrae el *payload* de datos para enviar al AS. Los datos son cifrados punto a punto entre ED y AS con la AppSKey.

Observar que el control de integridad finaliza en el NS, por lo que cualquier entidad capaz de manipular la comunicación luego de este punto, podría alterar el contenido de los datos en tránsito y su integridad ya no será verificada a nivel de LoRaWAN (independientemente de la corrección de la modificación). También se destaca que técnicamente el NS posee los datos necesarios para derivar ambas NwkSKey y AppSKey (ver 2.3.3.3), implicando control total sobre confidencialidad e integridad, aunque el AS es el verdadero destinatario de la AppSKey. Esta inconsistencia no es tratada en la especificación y por eso el NS **se asume confiable**. Para aplicaciones que requieran protección de confidencialidad e integridad total se recomienda utilizar una solución adicional de seguridad punto a punto. Esto escapa a la especificación de LoRaWAN [21].

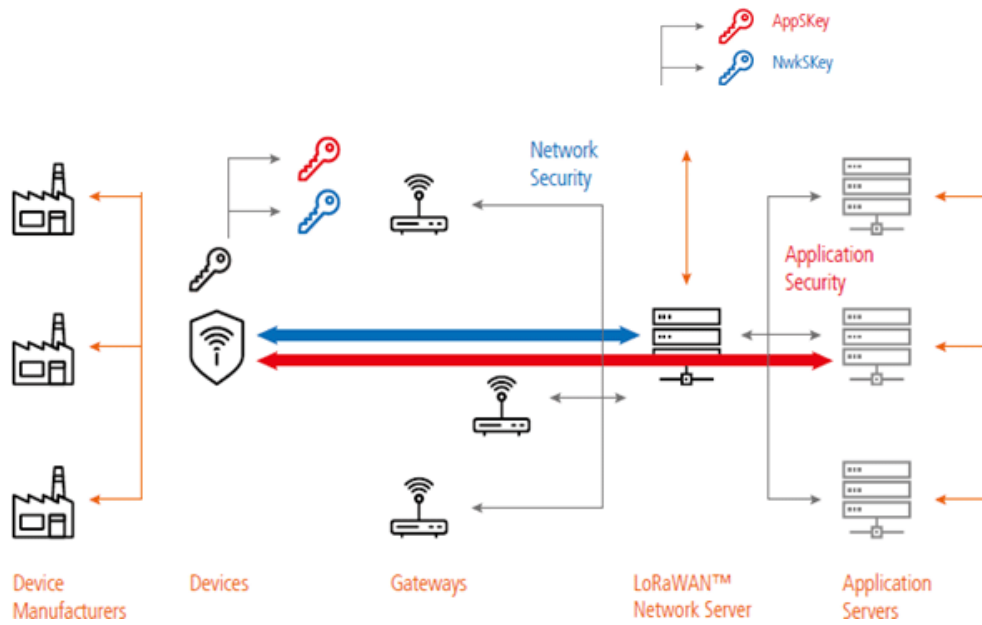


Figura 4: Esquema de seguridad en LoRaWAN 1.0.3.¹

2.3.3.2 Contadores

El protocolo define contadores de *uplink* y *downlink* de 16 o 32 bits. Una de las funciones de seguridad más importantes de estos es prevenir la repetición de mensajes previamente capturados (ataques de *replay*) [22]. LoRaWAN requiere que ambos NS y ED rechacen mensajes que contengan un *Frame Counter* (**FCnt**) menor al esperado. La especificación define además un mecanismo de confirmación de mensajes (*acknowledgement*, **ACK**) aunque es opcional.

Un detalle interesante es que para el caso de FCnt de 32 bits, no se ocupan más de 16 bits en el cabezal del mensaje, tomando los 16 bits menos significativos del contador [21], por lo que ambos NS y ED deben mantener referencia de los 16 bits restantes. Adelantándose a los detalles de integridad, aunque varios mensajes tengan el mismo valor de FCnt para un mismo *payload*, el código para validar la integridad va a ser distinto, pues su cálculo se basa en el valor completo de 32 bits del FCnt.

2.3.3.3 Activación

LoRaWAN provee dos métodos que permiten la activación y comunicación con el NS para un ED: activación por personalización (*Activation By Personalization*, **ABP**) y activación por aire (*Over-The-Air Activation*, **OTAA**). ABP implica que ambas claves de sesión (AppSKey y NwkSKey) y la dirección del ED en la red (*Device Address*, **DevAddr**) se graban en el *firmware* del ED por parte del fabricante y no son modificadas en su ciclo de vida. OTAA requiere una **AppKey** (clave raíz para cifrado AES de 128 bits) así como la información de identificación en ambos ED y NS, utilizadas en el mecanismo de *join* (unión) para derivar las claves de sesión. Como se ve en la Figura 5, el mecanismo de *join* (llamado *Join Procedure*) comienza con un

¹Figura oficial de LoRa Alliance, ya no disponible públicamente desde su sitio web.

mensaje *Join-Request* (**JR**) desde el ED al NS cuyos datos son **firmados** con la AppKey. El NS luego responde con un mensaje *Join-Accept* (**JA**) **firmado y cifrado** con la AppKey. Se destaca que las claves de sesión (ABP y OTAA) y AppKey (OTAA) deben ser únicas² para cada ED.

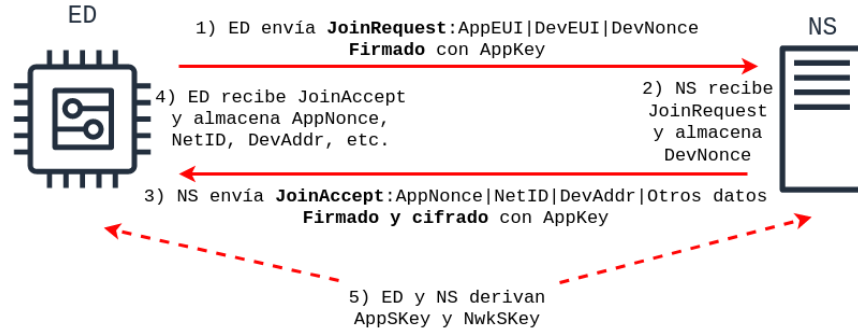


Figura 5: *Join Procedure* en LoRaWAN 1.0.3.

De la Figura 5 se observa que los campos relevantes en el intercambio de *Join Procedure* son:

- En el JR (enviado en texto plano):
 - *Application Identifier* (**AppEUI**): ID global de aplicación en el espacio de direcciones de IEEE EUI64 que identifica de forma única a la entidad habilitada a procesar el JR.
 - *End Device Identifier* (**DevEUI**): ID global de ED en el espacio de direcciones de IEEE EUI64 que identifica de forma única a un dispositivo. Sin embargo, se ha visto que tal unicidad no siempre se respeta en las implementaciones y que el valor es susceptible a *spoofing* [22].
- En el JA (cifrado con AppKey):
 - *End Device Address* (**DevAddr**): Identificador de 32 bits para el ED en la red LoRaWAN actual. El DevAddr puede verse como un identificador de sesión que usualmente cambia de sesión a sesión (luego de un *Join Procedure*), dependiendo de la implementación de la red. En dispositivos ABP el valor de DevAddr no cambia, al igual que sus claves de sesión.
 - *Network Identifier* (**NetID**): Valor compartido por todos los EDs en la misma red LoRaWAN.
- **DevNonce** (JR) y **AppNonce** (JA) son números/nonces pseudoaleatorios utilizados en la generación de claves de sesión para evitar ataques de *replay*.

Más específicamente, el código de verificación de integridad del JR se llama **MIC** (*Message Integrity Code*) cuya definición consiste en calcular otro código denominado **CMAC** (*Cipher-based Message Authentication Code*) y luego se extraen los primeros 4 bytes, sus fórmulas son las siguientes:

²La especificación de LoRaWAN lo recomienda, pero no obliga la unicidad de las claves. Si se repiten, vulnerar un ED en este sentido implicaría vulnerar otros.

```
cmac = aes128_cmac(AppKey, MHDR | AppEUI | DevEUI | DevNonce)
MIC = cmac[0..3]
```

El campo **MHDR** representa el cabezal MAC (*Medium Access Control*) del mensaje que posee el tipo de mensaje LoRaWAN y su versión³. Por otro lado, el MIC para el JA se calcula de forma casi análoga:

```
cmac = aes128_cmac(AppKey, MHDR|AppNonce|NetID|DevAddr|DLSettings|RxDelay|CFList)
MIC = cmac[0..3]
```

El campo **DLSettings** contiene parámetros de configuración para *downlink*, **RxDelay** es un *delay* entre transmisión (TX) y recepción (RX) y **CFList** es una lista opcional de frecuencias adicionales para comunicarse.

El mensaje JA es cifrado con AES-ECB⁴ usando la AppKey siguiendo la fórmula⁵:

```
aes128_decrypt(AppKey, AppNonce | NetID | DevAddr | DLSettings | RxDelay | CFList | MIC)
```

Finalizado el intercambio de mensajes, ambos ED y NS deben calcular las claves de sesión siguiendo las fórmulas⁶:

```
NwkSKey = aes128_encrypt(AppKey, 0x01 | AppNonce | NetID | DevNonce | pad16)
AppSKey = aes128_encrypt(AppKey, 0x02 | AppNonce | NetID | DevNonce | pad16)
```

Por especificación de LoRaWAN, no se da respuesta a un ED si el JR no fue aceptado. Lo usual es que este vuelva a intentar la unión a la red. Un detalle importante es que el NS cuenta con un mecanismo limitado de protección *anti-replay* de JR que consiste en almacenar los últimos N_D DevNonces aceptados en el *Join Procedure* y verificar en esta lista si los nuevos JR son replicados. En [23] se investiga la seguridad del *Join Procedure* y matemáticamente se analiza el costo-beneficio entre el tiempo que demora un atacante en poder repetir un DevNonce con un JR viejo y la probabilidad de que el ED repita un DevNonce naturalmente, teniendo como factor común a N_D . Dentro de sus conclusiones, se recomienda que N_D sea mayor al número aproximado de DevNonces que un ED genere en su vida útil y que no se implementen bloqueos a EDs que los repitan dado que puede ser aún más perjudicial bajo ataques DoS. La sesión queda confirmada en el NS **cuando recibe el primer mensaje *uplink*** con los parámetros de la nueva sesión.

³Específicamente el campo es *Major Version*, que representa el formato del paquete en la versión de LoRaWAN en la que fue codificado.

⁴ECB: *electronic codebook*, uno de los modos de cifrado en bloques.

⁵La operación es en realidad un descifrado, para que el ED solo implemente la función de cifrado [21].

⁶La función pad₁₆ agrega octetos de ceros hasta que el largo de los datos es múltiplo de 16.

2.3.3.4 Confidencialidad

La confidencialidad en LoRaWAN se consigue cifrando únicamente el *payload* de datos (*Frame Payload* o **FRMPayload** en la Figura 6), que son los datos intercambiados entre ED y AS. El resto de los cabezales de la Figura 6 como *MAC Header* (**MHDR**), *Frame Header* (**FHDR**) con sus campos de datos y el MIC son enviados sin cifrar.

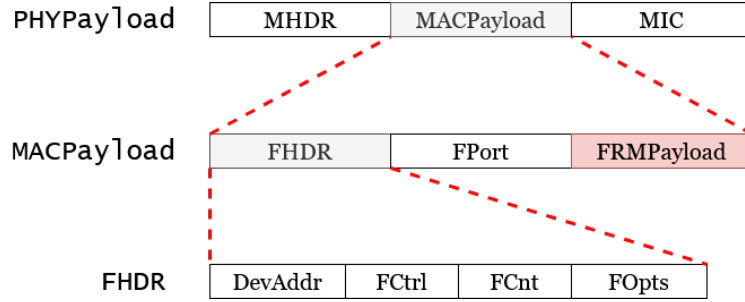


Figura 6: Desglose de PHYPayload hasta FHDR.

Como ya se mencionó, el cifrado AES-128 se hace con la AppSKey, en el modo CTR⁷, aunque si el valor de FPort es 0 se utiliza la NwkSKey (es un caso particular para comandos MAC donde no hay datos involucrados). Para el cifrado de datos se produce un *keystream* (S) de la siguiente manera:

$$\begin{aligned}
 i &= 1..k / k = \text{ceil}(\text{len}(\text{FRMPayload})/16) \\
 A_i &= (0x01 \mid (0x00 * 4) \mid \text{Dir} \mid \text{DevAddr} \mid \text{FCntUp o FCntDown} \mid 0x00 \mid i) \\
 S_i &= \text{aes128_encrypt}(K, A_i) \\
 S &= S_1 \mid S_2 \mid .. \mid S_k
 \end{aligned}$$

El *keystream* incluye el valor de FCntUp o FCntDown, lo que significa que el *keystream* nunca debería repetirse durante la sesión. Luego es aplicado XOR entre los bytes de FRMPayload y el *keystream* **para cifrar o descifrar** los datos (el proceso es idéntico en ambos casos y siempre se utiliza **encrypt** en el modo CTR para construir el *keystream*). El resultado final se obtiene de truncar hasta el largo original de FRMPayload.

2.3.3.5 Integridad

Para proteger la integridad de los mensajes en LoRaWAN, se utiliza un MIC similar a los ya mencionados. El MIC depende de todo el mensaje LoRaWAN y se agrega al final de cada mensaje. Como en 2.3.3.3, el MIC son los primeros 4 bytes del resultado de una operación AES-CMAC. En este caso la fórmula es:

$$\begin{aligned}
 \text{Msg} &= \text{MHDR} \mid \text{FHDR} \mid \text{FPort} \mid \text{FRMPayload} \\
 B_0 &= (0x49 \mid 4 * 0x00 \mid \text{Dir} \mid \text{DevAddr} \mid \text{FCntUp o FCntDown} \mid 0x00 \mid \text{len}(\text{Msg}))
 \end{aligned}$$

⁷CTR - *Counter Mode Encryption*. Es un modo de operación del algoritmo AES que se basa en un contador para cifrar *streams* o flujos de datos.

```
cmac = aes128_cmac(NwkSKey, B0 | Msg)
MIC = cmac[0..3]
```

B_0 es un arreglo de bytes compuesto por ciertos bytes fijos y otros variables, como **Dir** que es la dirección del mensaje (0 para *uplink* y 1 para *downlink*), DevAddr, FCnt con su valor correspondiente de *uplink* o *downlink* y el largo del mensaje (**Msg**).

3. Modelado de amenazas

3.1. Caso de estudio y metodología

Gran parte de este proyecto involucra generar un modelo de amenazas. Sin embargo, no es posible modelar sin un alcance de realidad definido. La realidad acordada en el marco de trabajo con la Intendencia de Montevideo es un despliegue LoRaWAN con sensores que envían sus medidas. Dada la generalidad del caso, se estudian las amenazas contemplando las potenciales configuraciones y ambos métodos de activación (OTAA y ABP). Además, se acordó investigar la familia de versiones 1.0.* de LoRaWAN que ya cubren la mayoría del espectro de versiones (solo dejando fuera la 1.1).

La metodología elegida sigue los lineamientos de OWASP [24, 25] para modelado de amenazas tomando lo que aplica a la realidad acordada y ajustándose a un buen ejemplo de modelado en IoT, particularmente sobre una infraestructura AMI (*Advanced Metering Infrastructure*) [26], el componente más visible de un *smart grid*. Es un modelado bien logrado y su estudio fue sugerido por los responsables del proyecto. El orden de los pasos seguidos en este modelado de amenazas se refleja en las siguientes secciones del informe hasta que el trabajo es concluido y también es listado en la sección 1.

Un paso extra agregado a la metodología es el análisis probabilístico de árboles de ataque con la herramienta SHARPE (ver 3.5.3), que no forma parte de la metodología propuesta por OWASP ni del *paper* [26]. Esto fue sugerido por los supervisores habiendo ya sido aplicado en el proyecto de seguridad de *smart grids* con UTE.

3.2. Diagramas de flujo de datos

Luego de comprendida la arquitectura y los aspectos de comunicación entre los componentes de LoRaWAN, basado en las recomendaciones de OWASP [24] se diagramaron los procesos involucrados y sus flujos de datos necesarios para lograr identificar y enfocarse en los elementos que son potencialmente amenazados. Para representar estos diagramas llamados diagramas de flujos de datos (**DFD**) se investigaron primero los ejemplos y notación de OWASP *Threat Model Cookbook* [25]. En segundo lugar, se analizó un complejo DFD correspondiente a un medidor maestro en [26] donde el desarrollo se da en el “interior” del dispositivo y no en visión macro como los ejemplos de [25]. Aunque se tomaron ambos enfoques, el más útil fue analizar los procesos internos.

Un DFD puede tener más de una notación, la que aquí se utiliza se representa con los siguientes elementos:

- **Entidad externa:** Puede representar un ser humano, un sistema o un subsistema. Dependiendo la granularidad del análisis puede ser más confuso identificarla como externa. Figura como un rectángulo.
- **Proceso:** Es una actividad o función donde se transforman los datos y su flujo, al transformarlos, ordenarlos o cambiando la dirección del flujo. Figura como un círculo.

- **Almacén de datos:** Contiene la información para su uso posterior. Puede ser una base de datos, un archivo o cualquier forma de almacenamiento. Figura como un rectángulo únicamente con bordes inferior y superior.
- **Flujo de datos:** Representa el flujo de la información. Determina el itinerario que toma la información desde las entidades externas a través de los diversos procesos y almacenes de datos. Figura como una flecha.
- **Barrera de confianza o interfaz:** Comúnmente llamada *trust boundary*, representa los límites en el flujo donde la confianza cambia o se pierde. Mientras se esté dentro de las barreras de confianza, se confía en que los datos y sus flujos no son comprometidos. Figura como una línea punteada.
- **Proceso complejo:** Es un elemento que puede verse de forma macro como un proceso en el que hay más procesos internos, figuren o no en el diagrama. Figura como un círculo de borde doble.

El DFD inicial (Figura 7) es general y da una visión holística de los flujos de datos en LoRaWAN hasta el punto en que estos son enviados a un proceso complejo externo.

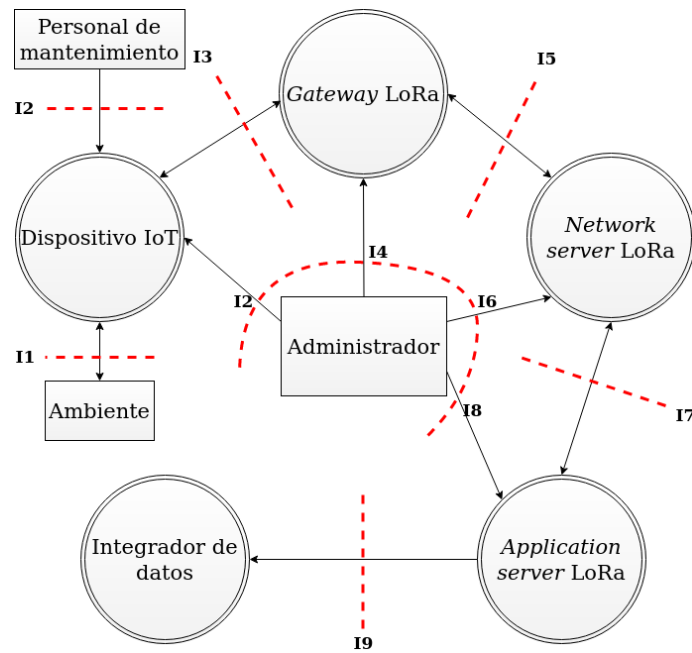


Figura 7: DFD general de LoRaWAN 1.0.3 hasta el envío de datos a una integración.

En este punto se identifican todos los procesos complejos, las entidades externas humanas (administrador y personal de mantenimiento) y no humanas (ambiente), los flujos entre los elementos y las interfaces. Estas últimas se identificaron basándose en la Tabla 1. Notar que a excepción del proceso complejo “Integrador de datos”, el resto está asociado a un elemento de la arquitectura LoRaWAN: Dispositivo IoT (ED), *Gateway* (GW), *Network server* (NS) y *Application server*

(AS).

<i>ID</i>	<i>Dónde</i>	<i>Comunicación</i>
I1	ED - Ambiente	Obtención de medidas; Función de influencia en el ambiente
I2	ED - Personal de mantenimiento/Administrador	Actualizaciones de <i>firmware</i> ; Gestión de claves; Configuración de parámetros; Consulta de logs
I3	ED - GW	Unión de dispositivo a la red; Envío y recepción de parámetros de cálculo de claves; Mensajes de control y datos
I4	GW - Administrador	Actualizaciones de <i>firmware/software</i> ; Configuración de parámetros; Consulta de logs
I5	GW - NS	Unión de dispositivo a la red; Envío y recepción de parámetros de cálculo de claves; Mensajes de control y datos
I6	NS - Administrador	Actualizaciones de <i>software</i> ; Gestión de claves; Configuración de parámetros; Consulta de logs
I7	NS - AS	Envío de claves; Mensajes de gestión y datos
I8	AS - Administrador	Actualizaciones de <i>software</i> ; Gestión de aplicaciones y dispositivos; Configuración de parámetros; Consulta de logs
I9	AS - Integrador de datos	Datos procesados

Tabla 1: Interfaces de los DFD con sus identificadores, extremos y qué fluye a través de estas.

El siguiente paso fue crear los DFD para el ED, GW, NS y AS (Figuras 8, 9, 10, 11 respectivamente). No se prosiguió luego del AS porque las integraciones son heterogéneas y no hay una estandarización de cómo deben tratarse los datos a partir de allí. Además, escapa al análisis de seguridad en el contexto de LoRaWAN. La construcción se dio recorriendo cada proceso de las comunicaciones en LoRaWAN y diagramando los flujos de estos, que eventualmente se tornaron más complicados de plasmar en un diagrama legible.

Es visible que los DFD resultantes tienen alto nivel de granularidad y complejidad. Esto sucedió porque se apuntó a tener flujos precisos y aprender en profundidad sobre los procesos durante la construcción de los diagramas. En cada DFD se muestra a los otros elementos como entidades externas porque se consideró que, para un elemento particular, el resto son externos detrás de una interfaz, más allá de que en sí pueden considerarse como procesos complejos en lugar de entidades (fue una convención de diagramación). Sucede algo similar con las aplicaciones que ejecutan en cada componente LoRaWAN (por ejemplo, la aplicación del ED) dado que son cajas negras por sí mismas en este análisis, pueden variar mucho según la implementación y consecuentemente sus procesos y flujos de datos.

A continuación, se explica el DFD del dispositivo IoT recorriendo sus flujos de datos y asociándolos a los procesos que se llevan a cabo en el dispositivo. Comprender este DFD es de suma importancia para entender las amenazas que se identifican posteriormente. Los otros DFD se construyeron de forma análoga, analizando los procesos en LoRaWAN y el flujo de información entre entidades.

En la Figura 8 hay tres interfaces o barreras de confianza. Partiendo por la interacción con el ambiente, el ED percibe de éste algún tipo de estímulo que se transforma en una medida, un dato a procesar. No es el caso en este trabajo, pero también puede darse la interacción opuesta si el ED es un “actuador” y ejerce una acción que influye en el ambiente (por ejemplo, prender una luz o encender un motor). Una aplicación en el ED gestiona entre otras cosas los datos percibidos.

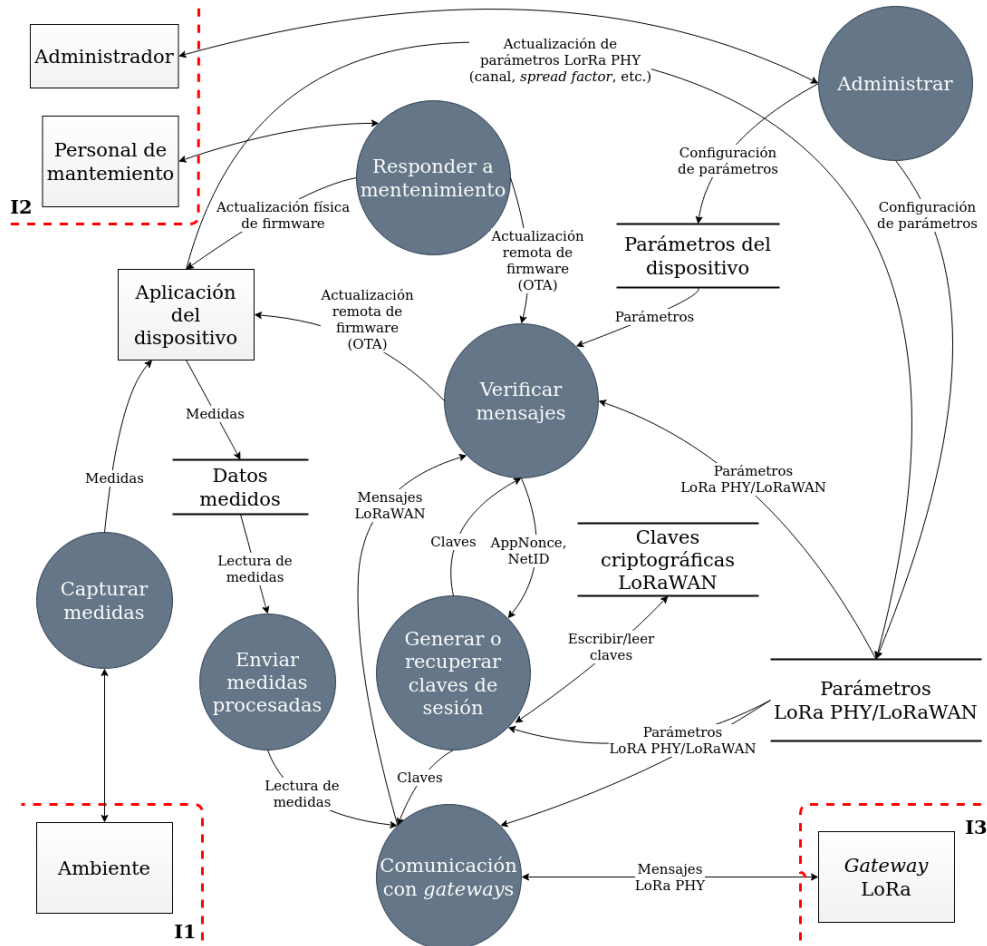


Figura 8: DFD de dispositivo IoT que reporta medidas del ambiente.

La aplicación aquí es considerada el *firmware* del ED, que puede ser actualizado física o remotamente según las capacidades del ED. El personal de mantenimiento a través de la interfaz que corresponde a administración/mantenimiento, carga el *firmware* y para el caso remoto debería darse un control extra de verificación en la recepción del contenido (para más detalles sobre actualizaciones OTA, ver 5.2.5). En lo que respecta al administrador, su función general es configurar el ED, ya sea para parámetros de funcionamiento interno o parámetros LoRa/LoRaWAN que afectan la comunicación a nivel de radiofrecuencia (por ejemplo, frecuencias o intervalos de recepción) o del protocolo subyacente (como claves o contadores). Estos parámetros se utilizan en la verificación y armado de mensajes.

Volviendo a la aplicación del ED, el nuevo dato puede almacenarlo en un registro a modo de

encolar para futuro envío, o en algún *buffer* para proceder a enviarlo inmediatamente. El dato es recuperado para un posible procesamiento, dándole formato para ser incluido como *payload* de un mensaje LoRaWAN. Luego de esto (en proceso de comunicación con GWs), es necesario obtener las claves de sesión almacenadas para cifrar el *payload* y generar el MIC. Además, se leen los datos de configuración LoRa que involucran el envío por el medio físico. Luego de armado el paquete y configurada la transmisión, se envía el mensaje al GW atravesando la barrera de confianza I3.

Por otro lado, mensajes llegan, representando distintas cosas según el método de activación del ED, además de los mensajes de datos. En el caso de OTAA, podría llegar un JA como respuesta a un JR enviado por el ED. Aquí el mensaje pasa por un proceso de verificación de integridad y autenticidad, precisando para ello recuperar la AppKey almacenada. Una vez verificado, se extrae el AppNonce y NetID para luego generar las nuevas claves de sesión y almacenarlas. Los mensajes también podrían ser comandos LoRa que cambian parámetros de transmisión/recepción, en cuyo caso estos parámetros serán almacenados para uso futuro. Aquí no se contemplan comandos a nivel de datos recibidos en *payload* LoRaWAN de *downlink*, pero la modificación a este DFD sería agregar un flujo desde el proceso “Verificar mensajes” a la entidad “Aplicación del dispositivo”, dado que es la encargada de procesar el comando y actuar acorde a este.

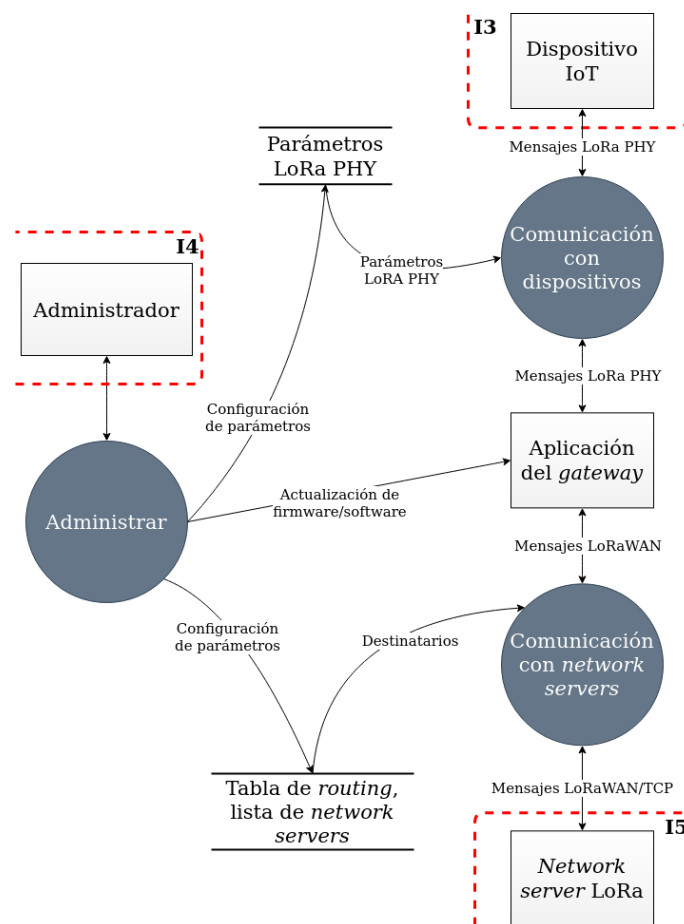
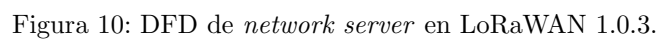


Figura 9: DFD de *gateway* LoRaWAN.



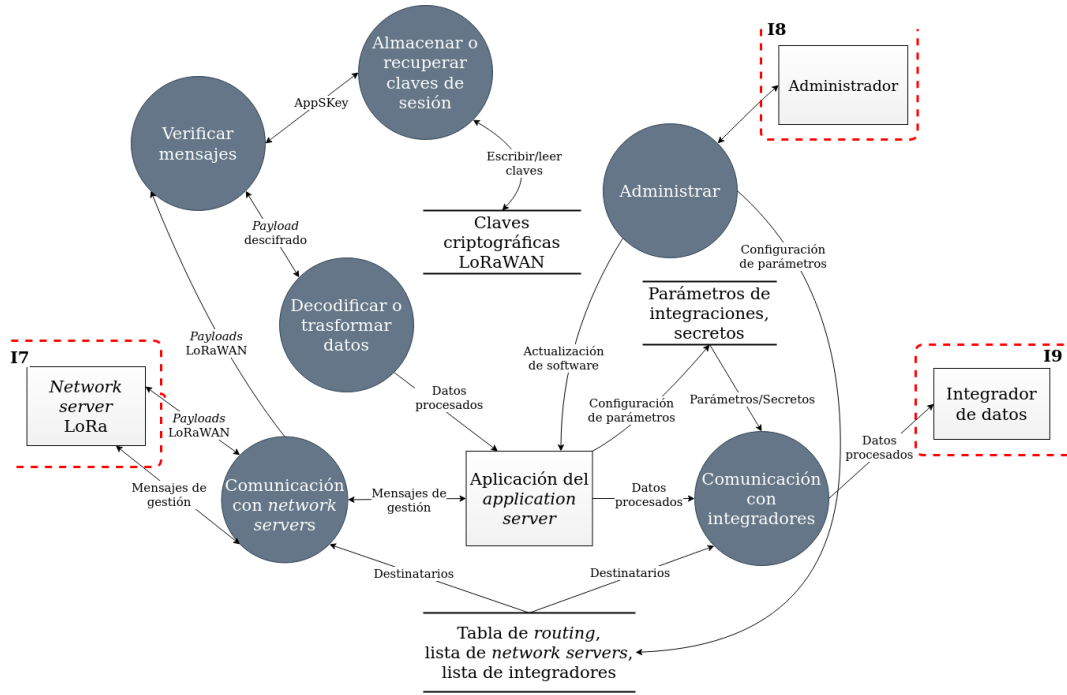


Figura 11: DFD de *application server* en LoRaWAN 1.0.3.

3.3. Identificación de amenazas (STRIDE)

OWASP sugiere identificar las amenazas siguiendo modelos estándar contemplando los potenciales actores de amenaza. Respecto a los actores, se hizo el análisis de la Tabla 6 que se encuentra en el Anexo A. Este cuenta con descripciones de los actores, sus objetivos usuales, la probabilidad estimada de encontrarlos en el escenario de LoRaWAN en la realidad que se está analizando y la prioridad asignada a contemplarlos en el modelado de amenazas. La probabilidad de acción es relativa entre los actores (aplica también a la prioridad), es decir que, más allá de los valores de probabilidad seleccionados⁸, unos siempre serán mayores que otros. Por ejemplo, un empleado administrador de servicios en infraestructura LoRa indiscutiblemente tendrá mayor probabilidad que un ente de control y por esto, la presencia de un actor cambia el balance de probabilidades/prioridades.

Para el modelo, se decidió seguir STRIDE [5] que propone preguntarse cómo acceder a ciertos recursos o interponerse en los procesos y qué sucede si se logra. Pueden utilizarse las letras de su acrónimo para identificar amenazas desde el inicio o se categorizan una vez que se obtiene un listado. Las letras de STRIDE apuntan a distintas propiedades de la seguridad y significan *Spoofing* (autenticidad), *Tampering* (integridad), *Repudiation* (no repudio), *Information disclosure* (confidencialidad), *Denial of service* (disponibilidad) y *Elevation of privilege* (autorización). La Tabla 2 muestra las amenazas resultantes con sus identificadores, nombres asociados y las interfaces por las cuales podrían darse.

⁸La asignación de probabilidades y prioridades seguramente sea distinta para otros investigadores.

<i>ID</i>	<i>Nombre</i>	<i>Interfaz</i>
<i>Spoofing</i>		
T01	Atacante autenticado en ED como personal de mantenimiento	I2
T02	ED falso	I3
T03	GW falso	I3
T04	Atacante autenticado en GW como personal de mantenimiento	I4
<i>Tampering</i>		
T05	Alterar comunicación entre EDs y GWs	I1, I3, I6, I7
T06	Mantenimiento local altera mediciones, configuración o <i>firmware</i> del ED	I2
T07	Mantenimiento local altera configuración o <i>firmware</i> de GW	I4
<i>Repudiation</i>		
T08	Modificación de <i>firmware</i> anónima en ED	I2, I3
T09	ED falso suplantando la identidad de un ED legítimo	I3
T10	ED niega haber recibido/enviado un mensaje	I3
T11	Modificación de <i>software/firmware</i> anónima en GW	I4
<i>Information disclosure</i>		
T12	ED filtra claves de sesión	I2
T13	ED filtra sus parámetros de configuración	I2
T14	ED filtra mediciones	I2
T15	Escucha de comunicación entre EDs y GWs	I3
T16	Escucha de comunicación entre GWs y NS	I4, I5
<i>Denial of service</i>		
T17	Abuso de medida para bloquear/ignorar medidas legítimas	I1
T18	ED no puede comunicarse	I1, I2, I3, I5, I6
T19	ED deja de cumplir funciones que afectan al ambiente	I2, I3
T20	Agotamiento de batería del ED	I3
T21	GW no puede comunicarse	I3, I4
T22	Sumidero de mensajes desde el GW (<i>sinkhole</i>)	I4
<i>Elevation of privileges</i>		
T23	Acceso remoto a ED	I2
T24	Acceso remoto a GW	I4

Tabla 2: Listado de amenazas siguiendo el modelo STRIDE.

Otras amenazas fueron listadas inicialmente, pero daban lugar a agruparlas en otra más general o eran demasiado específicas. Estas últimas se tomaron como referencia para los nodos en los árboles de ataque (ver 3.5.2). Una **importante aclaración** sobre la exhaustividad del listado: el alcance fue reducido a amenazas que involucraran ED y GW, dado que son los elementos que por sí mismos y su comunicación son poco conocidos y se cree que contribuye más al área de investigación. Sin embargo, se destaca que NS y AS son también cruciales (en especial NS) en el aseguramiento de LoRaWAN. Viendo al GW como un *forwarder* de mensajes, podría considerarse al NS como siguiente punto de comunicación con el ED y por esto inevitablemente se incluyen en el trabajo cuestiones de seguridad que lo involucran. De todas formas, si se desea profundizar en

NS y AS para futuros análisis se pueden utilizar como punto de partida los DFD, que sí cubren todos los componentes.

3.3.1. Spoofing

Abusando de la autenticidad está la posibilidad de que un atacante logre autenticarse en un ED como si fuera personal de mantenimiento (T01 y análogo en T04 para un GW), probablemente se requiera acceso físico al dispositivo, aunque podría llegar a ser remoto. También podría usarse un falso ED (T02) en el sentido de reemplazar parcial o completamente a un ED legítimo en la red, **no** en unir un ED malicioso pues esto requiere obtener parámetros (AppEUI y AppKey en caso de OTAA y claves de sesión en ABP) para la autenticación mutua, es decir, para lograr esto hay que tomar control del NS y AS (más complejo y fuera del alcance ya mencionado). Para la amenaza T03 sucede algo similar, pues para suplantar la identidad de un GW se requiere: por el lado del ED, conocer sus claves de sesión o abusar del *Join Procedure* y lograr enviar JAs maliciosos sabiendo otros parámetros (todo esto siempre y cuando haya un solo GW, pues anular la señal de más de uno complejiza el ataque). Por el lado del NS, se requiere conexión TCP/IP con este, algo mucho más complicado que también implica tener más control sobre la infraestructura.

3.3.2. Tampering

Para afectar la integridad, la amenaza más sofisticada pero poderosa es comprometer el canal ED-GW para alterar los mensajes (T05). Hay muchas formas de hacerlo (ver 3.5.2) y consecuentemente los requerimientos varían (acceso físico, acceso a las claves de sesión, comprometer el *hardware*, entre otros). Esta amenaza está vinculada con T02 y T03. Luego están T06 y T07 que apuntan a la alteración de datos, configuración o *firmware* por medio del acceso a mantenimiento local.

3.3.3. Repudiation

Dentro de las amenazas relativas al no repudio, todas se basan en la ausencia de un registro que permita identificar a un usuario o dispositivo. T08 y T11 son similares siendo que ambas logran una modificación de *firmware* y no hay forma de asociar ninguna identidad a este proceso. T09 es similar a T02 pero desde el punto de vista del no repudio. T10 se da cuando un ED niega haber recibido o enviado un mensaje. En la recepción, existe un mecanismo de confirmación de recepción (ACK) pero no es obligatorio para un ED implementarlo, por lo que si el ED no lleva un registro de mensajes (usualmente no lo hará porque sería un consumo excesivo de recursos) ni utiliza ACKs, no hay forma alguna de corroborar que el mensaje haya llegado al ED. En el envío, no hay más pruebas que la firma en el mensaje, que debe ser con la NwkSKey única para ese ED. Debería ser prueba suficiente pero existen formas de obtener esa clave y dependiendo de la configuración en el NS, podrían enviarse mensajes antiguos sin necesidad de conocerla (ataque de *replay*).

3.3.4. *Information disclosure*

Las amenazas T12-T14 son de filtración en el ED, el cual podría a través de una interfaz (por ejemplo, serial, con USB u otro tipo de conexión) revelar las claves de sesión o AppKey, parámetros de configuración relevantes o las medidas tomadas. Otras formas son almacenando en memoria no protegida estos valores y que un atacante logre realizar la lectura. Luego en T15, escuchar el canal es algo que cualquiera puede hacer en el espectro de LoRa, pero el canal cifrado no es legible sin las claves de sesión (excepto los JRs) que como ya se mencionó, podrían ser obtenidas. Lo mismo se da en T16, aunque para escuchar el canal en primer lugar debería estar comprometido el enlace TCP/IP.

3.3.5. *Denial of service*

La criticidad de la denegación de servicio es variable según la realidad, pero en el ambiente IoT y más en radiofrecuencia, esto puede ser relativamente simple de llevar a cabo. En el canal aéreo afectan técnicas de *jamming* [27] por ejemplo, generando interferencias que invalidan mensajes. Junto con cualquier otra técnica que averíe la transmisión/recepción, dan lugar a las amenazas T18 y T21. Luego se da la amenaza relativa a seguridad física y alimentación energética (T20, podría estar junto a T18 y T21). Recordando que se analizan EDs que toman medidas, podría afectarse físicamente el ambiente para que medidas legítimas no lleguen a tomarse (por ejemplo, con ruidos excesivamente altos a un medidor de sonido). Si el ED además tiene algún impacto en el ambiente (por ejemplo, una luminaria), podría afectarse para que no cumpla su función en este correctamente (apagar la luz, o emitir intermitentemente) en T19. Por último, en caso de comprometer un GW, con ataques como modificar la tabla de enrutamiento hacia el NS, los mensajes no se enviarían a ningún lado (T22).

3.3.6. *Elevation of privilege*

Para dispositivos como un ED o GW que no necesariamente tienen manejo de permisos con perfiles de usuario, se vuelve difuso distinguir una amenaza de escala de privilegios. Pero considerando que es un abuso de autorización, se decidió tomar los accesos remotos al ED y GW para listar en esta categoría (T23 y T24).

3.4. Identificación de activos

Con las amenazas identificadas, el siguiente paso son los activos. Para ello se recurrió a la siguiente definición [26, 28]:

Un activo es cualquier cosa de valor que "necesita protección". Puede involucrar información, flujos, procesos, dispositivos físicos e incluso conceptos intangibles como la reputación. Un activo podría estar compuesto de otros activos.

Investigando sobre las formas de relevar activos, una publicación cita a [28] donde se dio tal relevamiento, aunque aplicado a ingeniería de *software*. Se adaptó el enfoque, convocando a una reunión con los responsables e involucrados en el proyecto para listar y argumentar los potenciales activos. Así surgió la siguiente lista:

- Dispositivo IoT
- Datos medidos
- *Gateway*
- *Network server*
- Plataforma FIWARE
- Servicios *core* (red interna)

Esta lista está ordenada por prioridad, siendo el dispositivo IoT el más prioritario. Los activos que no son obvios son los últimos dos. La plataforma FIWARE (descrita resumidamente en [8]) es una solución que centraliza y recolecta datos de todos los EDs (estén o no en la red LoRaWAN). Posee componentes como un *context-broker* que sigue el patrón publicador/suscriptor (aplicaciones se suscriben a eventos de recepción de datos de los EDs), un almacén de datos para registro temporal, proveedor de identidad y otros. En caso de verse comprometida esta plataforma, el daño podría ser crítico. En cuanto a los servicios *core*, refiere a todos los servicios de la organización que son esenciales y en caso de comprometerlos, sucede lo mismo con el impacto. Pero como estos activos (a pesar de ser totalmente válidos) se escapan al alcance definido por el marco LoRaWAN, no fueron analizados en este proyecto. La exclusión del NS ya se mencionó en 3.3. Finalmente, siendo que el GW es menos probable de ser alcanzado físicamente en el caso de estudio (suele ser una antena difícil de alcanzar), que las vulnerabilidades del ED ya van a contemplar al GW en cierta medida (pues es el siguiente punto directo de comunicación) y que de no acotar los activos los siguientes pasos del análisis se vuelven más complejos y ramificados, para lo que sigue del análisis también se dejó fuera al GW.

3.5. Análisis de ataques

3.5.1. Objetivos de ataque

Ya identificados los activos y seleccionados los más críticos (ED y datos), el siguiente paso es deducir objetivos de ataque (*attack goals*) sobre estos. Desde una visión ofensiva, con mentalidad de un atacante surgieron los objetivos de ataque de la Tabla 3:

<i>ID</i>	<i>Activo</i>	<i>Objetivo de ataque</i>	<i>Descripción</i>	<i>Impacto en activo</i>	<i>Prioridad</i>
A01	ED	Modificación indebida de mensajes salientes	Los mensajes para ser enviados por el ED son modificados de forma que el mensaje recibido por el GW no es el original.	Integridad, Confidencialidad	Alta
A02	ED	Suplantación de dispositivo	A nivel de LoRaWAN se introduce un nuevo ED que anula uno legítimo, pero en la red figura como el original y envía/recibe mensajes como tal.	Integridad	Alta

Continuación de la Tabla 3 de la página anterior

A03	ED	Acceso administrativo al dispositivo	Se logra acceso al ED que permite modificar su configuración y leer la información que manipula.	Integridad, Disponibilidad, Confidencialidad	Alta
A04	ED	Alterar comportamiento de dispositivo	Se logra modificar lo que el ED hace, pudiendo tener una función completamente diferente a la original o no respetando reglas definidas por el protocolo o por el negocio.	Disponibilidad, Integridad	Alta
A05	Datos	Modificación indebida	Manipulación no autorizada de datos ya medidos en cualquier parte del flujo de LoRaWAN.	Integridad	Alta
A06	ED	Dejar fuera de la comunicación a dispositivo	El ED no puede comunicarse, ya sea por incapacidad de emitir mensajes, o incapacidad de recepción/interpretación de los mensajes en GW o NS.	Disponibilidad	Media
A07	Datos	Indisponibilización	Los datos no llegan a destino.	Disponibilidad	Media
A08	Datos	Afectar mediciones	Afectar la medida a nivel de ED o ambiente.	Integridad	Media
A09	ED	Geolocalizar dispositivo	Revelar posición geográfica de ED basándose en sus mensajes.	Confidencialidad	Media
A10	Datos	Lectura no autorizada	Lectura no autorizada de los datos ya medidos en cualquier parte del flujo de LoRaWAN.	Confidencialidad	Baja

Tabla 3: Objetivos de ataque para los dos activos críticos por orden de prioridad.

Cada objetivo de ataque fue asociado a un activo y se señalaron las propiedades de seguridad afectadas más importantes, esto no quita que en cierta (menor) medida se afecte otra propiedad básica o más propiedades. En general los objetivos que afectan la integridad ya son marcados con prioridad alta (A01-A05), con excepción de A08 de prioridad media. Otros objetivos de ataque que surgieron eran redundantes o serían incluidos en más de un objetivo de los ya listados, por lo que fueron considerados más un medio que un fin, siendo llamados subobjetivos de ataque y en 3.5.2 son sub-nodos de los árboles de ataque. Un ejemplo de esto es el subobjetivo “Obtener claves de sesión”. Por otro lado, hay objetivos como A02 y A03 que en sí mismos son objetivos de ataque válidos con propósitos distintos a los de objetivos que pertenecen como sub-nodos en términos de árbol de ataque.

Los objetivos de ataque A01-A06 tienen sus propios árboles de ataque, por lo que serán comentados en más detalle en 3.5.2. En cuanto al resto, se describen aquí. A07 es un objetivo relativamente genérico apuntando al dato medido y refiere a su indisponibilización, es decir, por

algún motivo no llega a destino. Esto podría darse como consecuencia del objetivo A06, dado que el ED es afectado, o por ataques referentes al ambiente, de forma que los datos no puedan ser medidos en absoluto y por lo tanto tampoco puedan ser enviados, lo cual es totalmente dependiente del tipo de medida y dispositivo. A08 es similar a A07 aunque no se busca afectar la disponibilidad, sino la integridad. La geolocalización de dispositivos en LoRaWAN se busca explotar en A09, problema común a las tecnologías inalámbricas basándose en intensidad de las señales y el tiempo en aire de los mensajes. Tanto ED como GW podrían ser ubicados por triangulación con otros dispositivos que recepcionen las señales del entorno, lo que requiere precisión, equipamiento y movilidad. Sin embargo, si la red soporta EDs clase B, el GW podría enviar *beacons* con coordenadas de sus antenas donde estos por especificación no se cifran ni firman [21]. A10 es la lectura de datos sin autorización, que podría darse en el ED en interfaces de *debug* o escuchando el tráfico si se tienen las claves de sesión. De hecho, sin tener las claves se debe recurrir a ataques externos a LoRaWAN. Para el caso de estudio, la sensibilidad de los datos en términos de confidencialidad no era tan relevante comparada con las otras propiedades de seguridad y por esto tiene baja prioridad, sin embargo, en otras realidades la lectura de datos puede ser crítica. Aunque no se abarque puntualmente A10, los ataques a analizar que afectan la integridad en general ya cubren también la confidencialidad.

3.5.2. Árboles de ataque

El siguiente paso en la metodología elegida es modelar los objetivos de ataque de más prioridad obteniendo como salida sus correspondientes *árboles de ataque*. Estos modelan cómo un atacante podría cumplir sus objetivos y dan visibilidad de forma gráfica de qué medidas de seguridad podrían tomarse para mitigar o eliminar amenazas. Siguiendo ejemplos de modelos de OWASP *Threat Model Cookbook* [25] y teniendo en mente los objetivos de prioridad alta de la Tabla 3 se crearon los árboles de las siguientes secciones. A06 también tiene un árbol de ataque pues a pesar de tener prioridad media, la viabilidad de afectar la disponibilidad es tan alta en la teoría que se decidió dedicarle un lugar en el estudio.

3.5.2.1 A01, A02 - Modificación indebida de mensajes salientes y suplantación de dispositivo

La Figura 12 muestra el primer árbol de ataque modelado. A simple vista es medianamente complejo, pero lo es más aún pues en este (y los siguientes árboles) la presencia de un nodo con borde “de nube” significa que dicho nodo es un subárbol. En este caso, el nodo “Obtener claves de sesión” es el subárbol de la Figura 15. Por otro lado, el nodo “Suplantación de dispositivo” podría ser un nodo con subárbol, pero se decidió dejar ambos objetivos de ataque juntos pues comparten nodos y fue preferible denotar eso.

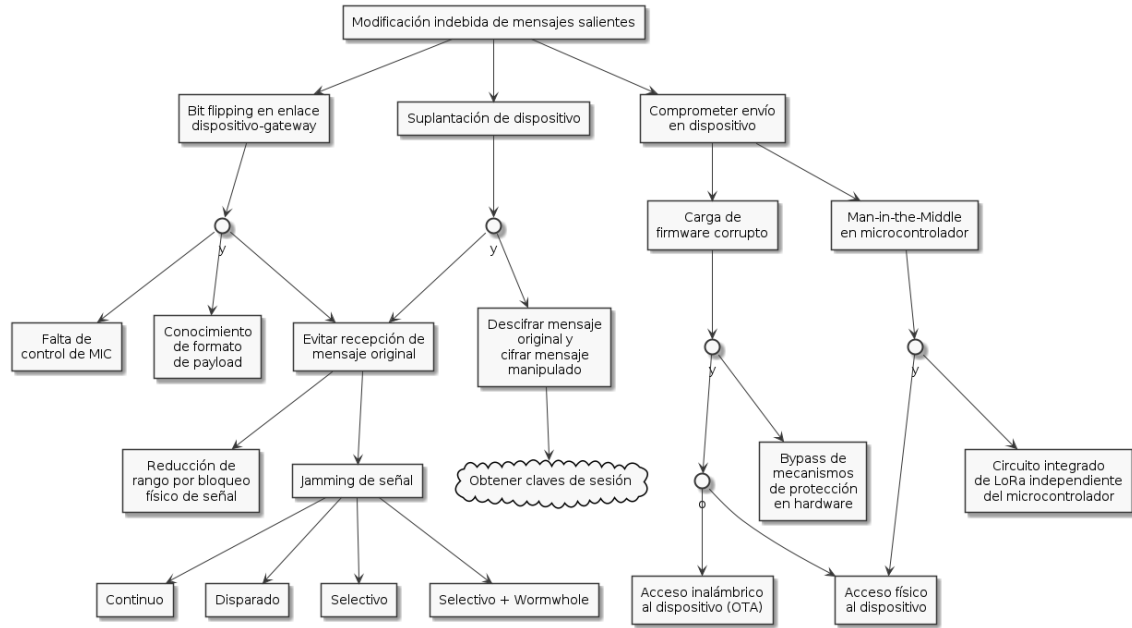


Figura 12: Árbol de ataque para el objetivo “Modificación indebida de mensajes salientes”, incluyendo también a “Suplantación de dispositivo”.

El nodo raíz es el objetivo de ataque inicial. Para conseguirlo, el atacante puede optar por las tres opciones **por separado** (esto representa un **OR** inclusivo lógico), es decir, para cumplir el objetivo se puede tomar cualquiera de los caminos y si uno no se puede implementar, no quita que el objetivo se pueda cumplir por los otros dos caminos.

De izquierda a derecha, la primera rama busca vulnerar la integridad en el enlace entre ED y GW mediante *bit-flipping*, en otras palabras, esto se da cambiando el valor de al menos un bit del mensaje. Podría emitirse señal en el mismo canal del mensaje original y que por interferencia se cambien los bits, pero este camino no garantiza que la modificación se dé y mucho menos que se cambien bits de una posición objetivo. Por lo tanto, este ataque tiene tres precondiciones: que la implementación del NS no controle la integridad (haga caso omiso al valor del MIC), conocimiento previo del formato del *payload* de aplicación y evitar que el mensaje original llegue a destino (pues la idea es enviar otro mensaje con el bit objetivo cambiado). Observar que las tres precondiciones deben cumplirse **a la vez** (esto representa un **AND** lógico) y se modela en el árbol como un nodo circular con el texto “y”. Un ejemplo de *bit-flipping* es el siguiente [29]:

Si el ED envía los siguientes datos en formato JSON como *payload*: {"ID": "34", "Temp": "24"}

Y el texto cifrado se ve así: 750F7F9B6366B4228172FB36FDBE51A3DCC1A85D463D70

Recordando el cálculo del cifrado de la sección 2.3.3.4, este último es generado con XOR entre el *payload* y el *keystream* sin mezclar ningún texto en el proceso [30]. Sabiendo cuáles son los bytes correspondientes al dato, en este ejemplo se suponen 5D46, puede modificarse 5D por 5A resul-

tando en el siguiente texto plano cuando se descifre: {"ID": "34", "Temp": "54"}. La viabilidad es mayor si se tiene la NwkSKey pues ya no sería necesario que la validación del MIC se omita para el éxito del ataque.

El nodo “Evitar recepción de mensaje original” es compartido por la rama de suplantar al ED. Las opciones para cumplir este subobjetivo son reducir el alcance de la señal del ED físicamente por bloqueos con ciertos materiales como con cualquier transmisión inalámbrica (por ejemplo, con una jaula de Faraday), u ocupar el canal del mensaje en cuestión para que haya interferencia (cambiando bits de este) y el mensaje se descarte por tener MIC inválido. Esta última técnica es llamada *jamming* y [27] la clasifica en cuatro tipos:

- Continuo: Constantemente se emite en el canal objetivo.
- *Triggered* o “disparado”: En el instante que se detecta señal en el canal, se transmite en este sin importar qué se esté intentando transmitir.
- Selectivo: Sigue al menos un criterio para iniciar el *jamming*. Primero se detecta el preámbulo del mensaje LoRa, luego se leen los bytes siguientes hasta determinar si el mensaje entra o no en el criterio. Ejemplos de criterios podrían ser: tener la condición de que haya un valor específico en los campos DevAddr (para afectar a un ED particular), según el valor del FCnt (por ejemplo, mayor o igual a cierto valor) o según el tipo de mensaje.
- Selectivo + *wormhole*: Es una combinación del anterior, pero sumándole un dispositivo a una distancia considerable (en lo posible cerca del GW) de forma que, el que está cerca del ED detecte que se debe iniciar el *jamming* y mande la señal por un canal con menor latencia al otro, entonces este último comienza el *jamming* antes de que el mensaje esté llegando o al menos con más probabilidad de interferir. El nombre es por el ataque *wormhole* o agujero de gusano, asociándolo a su concepto de dos puntos separados en el espacio, pero conectados por una especie de túnel.

Claramente el *jamming* selectivo es más sofisticado y efectivo que el continuo y *triggered*, pues estos últimos son muy fáciles de detectar (no se reciben mensajes de una frecuencia particular) y los administradores pueden marcar la frecuencia o todo el segmento de red como afectados, cambiar la frecuencia de las comunicaciones o activar *channel hopping* (saltar entre canales automáticamente). La Figura 13 muestra los períodos de cada etapa del *jamming* selectivo y *triggered* en el tiempo total de transmisión del mensaje.

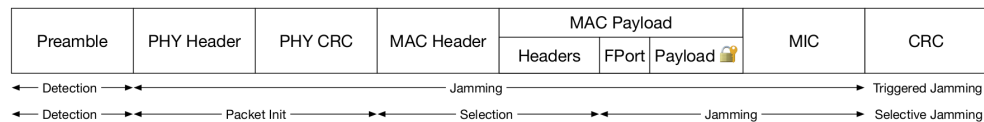


Figura 13: Etapas de *jamming* selectivo comparado a *triggered* [27].

En [31] se introduce otra clasificación que se explica gráficamente en la Figura 14. Aquí se introduce el tipo *Random* que es como el continuo, pero en intervalos no predecibles. También se suma el tipo *Deceptive*, donde en lugar de transmitir ruido, se transmiten mensajes LoRaWAN

válidos consecutivamente. Esta clasificación puede verse como variante de la primera así que para el árbol no se consideran, pero se debe tener presente su existencia.

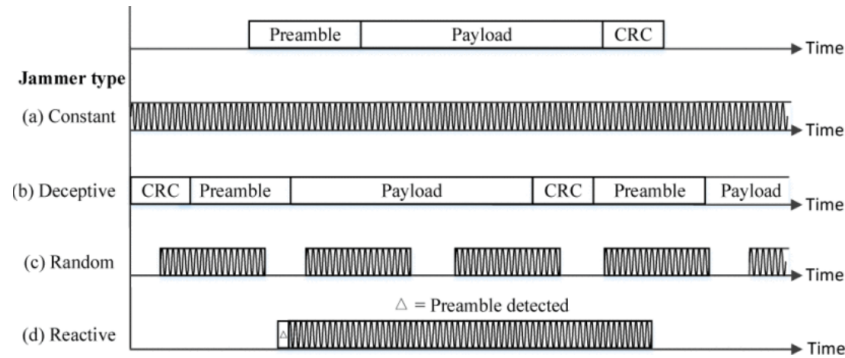


Figura 14: Clasificación alternativa de *jammings* [31].

Pasando a la segunda rama, se abarca el objetivo A02. En este se deben dar a la vez las precondiciones de evitar que se recepcione el mensaje original y que una vez este es capturado, sea descifrado, modificado, cifrado y enviado al GW. Para el proceso de descifrar/cifrar es estrictamente necesario obtener las claves de sesión y aquí es donde entra en escena el subárbol de la Figura 15. El cual se analizará luego de describir el árbol principal.

Para la tercera rama se busca que el envío no se comprometa en el canal LoRa, sino en el ED mismo. Para esto es posible ir por dos caminos. El primero es cargar *firmware* corrupto en el ED, ya sea el original modificado o uno distinto. Esto requiere por un lado saltar los mecanismos de protección activos en el *hardware* **si los tuviera**⁹, contando con gran experiencia técnica en electrónica y posiblemente contando con un *exploit*. Esto es dependiente del modelo del *hardware*, como se comentó en el caso de ESP32 en 2.2. El otro punto clave para cargar *firmware* es tener acceso al ED, físico o inalámbrico. En el caso de acceso físico se debe contar con el equipamiento necesario (puede ser tan solo un cable USB como también placas auxiliares, adaptadores y conectores de varios tipos). El acceso inalámbrico existe en *hardware* que soporta cargas de ese tipo, suelen llamarse Over-The-Air (OTA)¹⁰ y son útiles cuando el acceso al ED es poco práctico o inviable sin daño físico, por ejemplo, si el ED tuviera una carcasa sellada sin conectores de datos expuestos o tornillos. La segunda forma de comprometer el ED es con *Man-in-the-Middle* (**MitM**) entre el microcontrolador (**MCU**, *Microcontroller Unit*) y el transceptor LoRa. Este ataque descrito en [29] precisa claramente acceso físico al dispositivo y además el MCU debe estar en un circuito integrado independiente al del transceptor LoRa. Esto permite conectarse desde otro MCU directamente a los pines del transceptor o reemplazar completamente el MCU original y, si el modelo de transceptor lo permite, las claves LoRaWAN serán almacenadas y gestionadas en el transceptor. Esto quiere decir, que las claves no son necesarias para escribir/leer mensajes, sino que se envía cualquier dato al transceptor y este se encarga de cifrar el mensaje y enviarlo (análogo en la recepción).

⁹Es de suma importancia contar con protección del *hardware*, según las capacidades de este puede ser para lectura, escritura o ambas.

¹⁰No confundir OTA con OTAA de LoRaWAN.

Volviendo al subárbol de la Figura 15, hay cuatro formas básicas generales de obtener las claves de sesión. La primera es robarlas por filtraciones de datos en el proveedor o fabricante, la segunda es derivarlas de la AppKey (sólo en OTAA), la tercera es robarlas del ED y la cuarta es robarlas del NS. El último caso se escapa al análisis más concreto de la tecnología LoRaWAN y es un ataque de servidor tradicional en lo que respecta a su acceso y obtener secretos (en este caso las claves), por lo que no se ahonda en ello. Se precisaría acceso a la red interna donde esté ubicado (si estuviera expuesto a Internet esto deja de ser necesario) y luego se puede acceder por medio de algún *exploit* local o remoto, u obtener las credenciales de acceso al NS por distintas técnicas como *phishing*, fuerza bruta, ingeniería social, entre otras.

La filtración de datos en el proveedor refiere a hacerse de la información atacando o investigado al proveedor del ED, por filtraciones de datos públicas en Internet, hackeo a sus servidores, o que esté el código fuente en Internet y el valor de las claves por defecto no se cambie al desplegar la red.

En cuanto a derivar claves de sesión de la AppKey, primero esta debe conseguirse. Una forma es deducirla, dado que se conocen casos de AppKeys que siguen fórmulas como $AppKey = DevEUI + AppEUI$, $AppKey = AppEUI + DevEUI$, o hashes (completos o truncados de MD5, SHA-1, SHA-2, SHA-3) de estos resultados o sus operandos, entre otras posibles combinaciones [22]. Hay que recordar que DevEUI y AppEUI son valores que se transmiten sin cifrar en el *Join Procedure*. Para validar que la AppKey deducida es correcta se generan claves de sesión que permitan validar MICs de mensajes, descifrar *payloads* o en el caso más simple, con un par JR-JA lograr descifrar el JA ya que este se cifra con la AppKey. El proceso puede ser más complejo dependiendo con qué mensajes se cuente, si se trata de ABP solo es posible comparar con dos mensajes de datos enviados y aquí deben deducirse ambas claves de sesión, proceso que puede llevar poco tiempo si se tienen claves débiles (no generadas aleatoriamente y que sigan un patrón predecible como todo ceros o todo unos) o tiempos inviables con fuerza bruta para 2^{128} posibilidades por clave en el peor caso. Si la AppKey intenta robarse puede ser en el proveedor del ED (por filtraciones de datos) o robarse del ED.

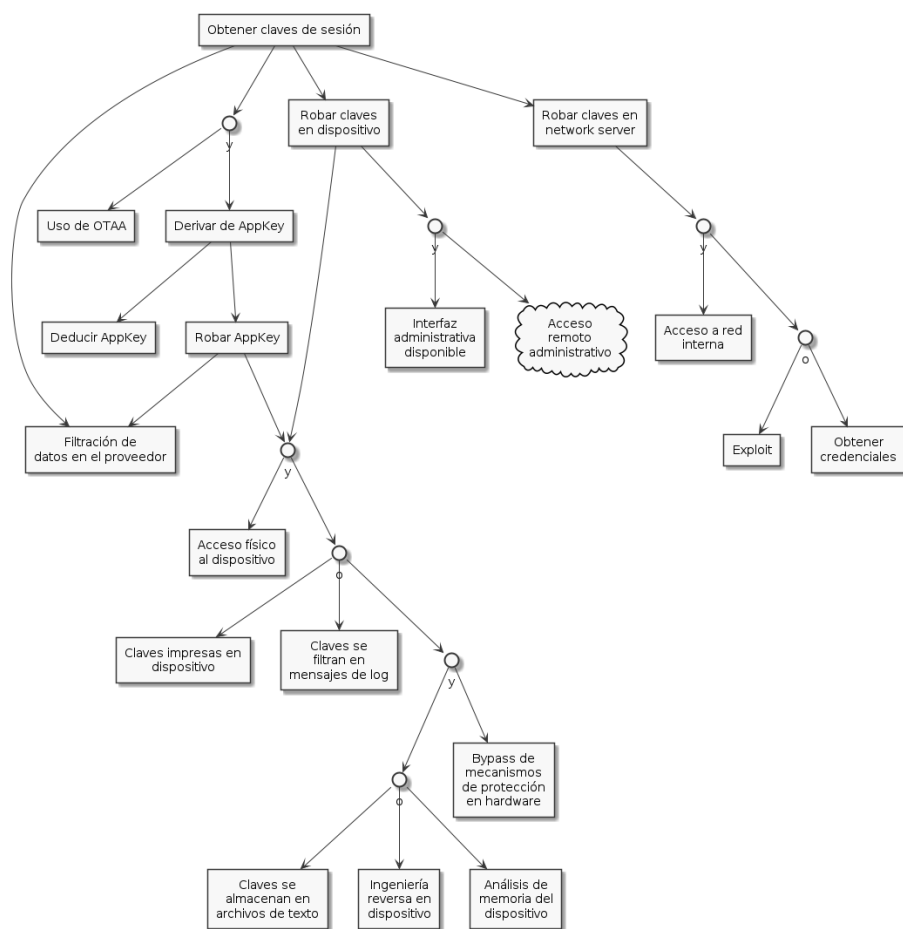


Figura 15: Árbol de ataque para el subobjetivo “Obtener claves de sesión”.

Robar claves del ED implica acceso físico a este, una vez obtenido hay varias formas de visualizarlas. La más simple es que figuren impresas, esto se menciona porque se ha visto [22] pero es muy poco probable. La siguiente en nivel facilidad es conectarse a una interfaz que permita ver logs de *debug* o informativos, que eventualmente en un *Join Procedure* se imprima en pantalla la AppKey o las claves de sesión cuando se generan. Luego siguen las técnicas que se basan en acceder al interior del ED que suele ser extrayendo *firmware*, memoria volátil o memoria no volátil (como una *flash* o un pequeño sistema de archivos). Todos estos terminan precisando saltarse los mecanismos de protección de *hardware* que ya se mencionaron en esta sección. Luego de la extracción puede haber archivos de texto con las claves en ellos, pueden figurar como *strings* en la memoria o analizando ejecutables con ingeniería reversa pueden estar como constantes o variables.

Finalmente, para la rama de robar las claves del dispositivo, se puede ir por la subrama descrita anteriormente o si se pudiera, acceder remotamente al ED en modo administrativo (subobjetivo explicado como parte del árbol de A03 en 3.5.2.2). De esta forma podría haber una interfaz de configuración con las claves.

A pesar de que los caminos de las hojas a la raíz son varios, es importante notar que hay nodos requeridos por más de un camino. El nodo “*Bypass* de mecanismos de protección en *hardware*” es dependido por **tres** caminos (sin contar los de sus ramas hermanas que multiplicarían la cantidad de ataques con este nodo como requerimiento). Con el mismo criterio, el nodo “Acceso físico al dispositivo” es dependido por **cuatro** caminos. Si la probabilidad de ejecutar estas hojas pasa a ser próxima a 0, esto reduce drásticamente la probabilidad total de cumplir con el objetivo de ataque raíz. Este tipo de análisis se hará en la sección 3.5.3.

3.5.2.2 A03 - Acceso administrativo al dispositivo

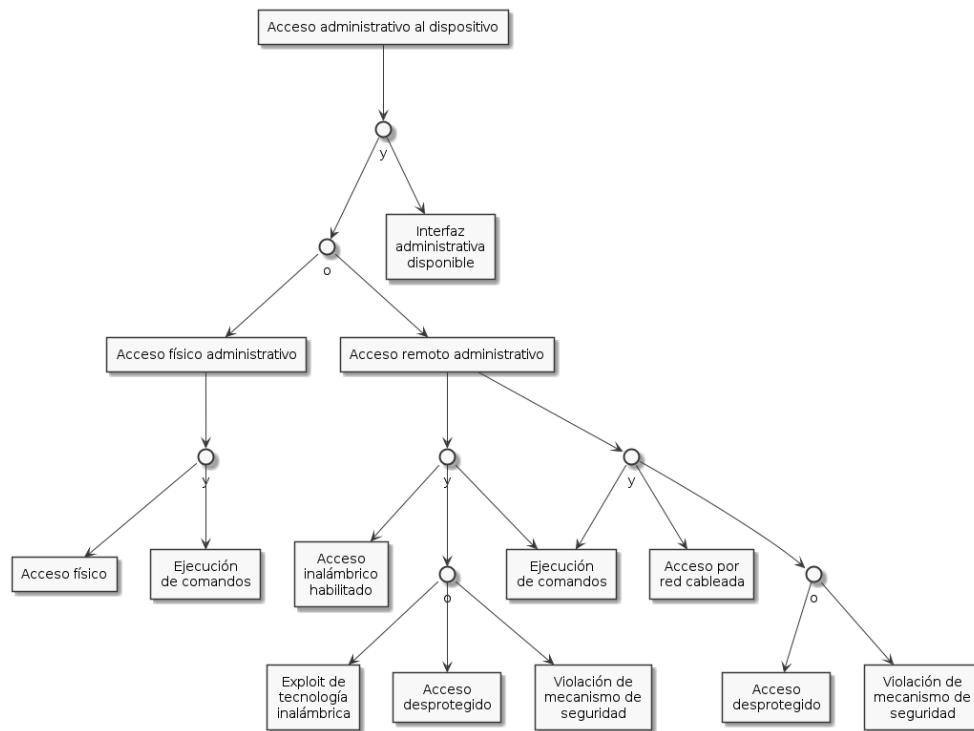


Figura 16: Árbol de ataque para el objetivo “Acceso administrativo al dispositivo”.

El acceso administrativo al dispositivo se basa exclusivamente en que haya una interfaz administrativa disponible. EDs sofisticados podrían llegar a tenerla para acceso remoto del proveedor si se expone a Internet o en caso de mantenimiento local si se envía a reparar o para configuración inicial. Esta disponibilidad se ve en el árbol de la Figura 16 inmediatamente luego de la raíz, sin ella no hay ataque. Luego puede darse por acceso físico o remoto según la naturaleza de la interfaz administrativa. En el primer caso es requerido acceso físico al ED y por medio de la conexión que este ofrezca, poder ejecutar comandos vía una interfaz gráfica que se comuniquen con algún protocolo propietario o mediante una línea de comandos (el ED podría tener un sistema operativo ligero con intérprete de comandos). En el caso remoto se distinguen los dos casos de remoto inalámbrico o cableado. Es una sutileza, pero se separó porque los nodos no son exactamente iguales.

El acceso inalámbrico requiere en primer lugar que este esté habilitado y debería ser por una tecnología diferente a LoRa como Wi-Fi o Bluetooth. Luego las posibilidades independientes son que no exista protección alguna en el acceso (acceso sin credenciales), que exista un *exploit* particular para el *hardware* y la tecnología inalámbrica permitiendo saltarse la seguridad, o violar el mecanismo de seguridad que el protocolo ofrece (como sucede con la paridad en BLE [3, 11]). Cumpliendo con cualquiera de estas opciones se requiere finalmente poder ejecutar comandos en la interfaz expuesta como ya se mencionó. El caso de acceso por red cableada es análogo pero aplicado a tecnologías no inalámbricas.

3.5.2.3 A04 - Alterar comportamiento de dispositivo

Este objetivo se analiza desde el punto de vista de cómo el ED debería funcionar, no necesariamente en LoRa, sino en su comportamiento general. Por dar ejemplos, un termostato podría marcar erróneamente la temperatura y afectar la calefacción, una luz podría apagarse o ser intermitente y una cerradura magnética podría deshabilitarse. Se identificaron tres ramas principales (Figura 17): cargar *firmware* corrupto, alterar algún aspecto del *hardware* y por mensajes maliciosos a nivel de datos. El primero ya fue explicado en 3.5.2.1 y en este objetivo pretende utilizarse para cargar un *firmware* que no haga absolutamente nada (caso más sencillo) o que, conociendo el *firmware* original y cómo se interactúa con el *hardware*, agregar/reemplazar lógica que afecte el comportamiento.

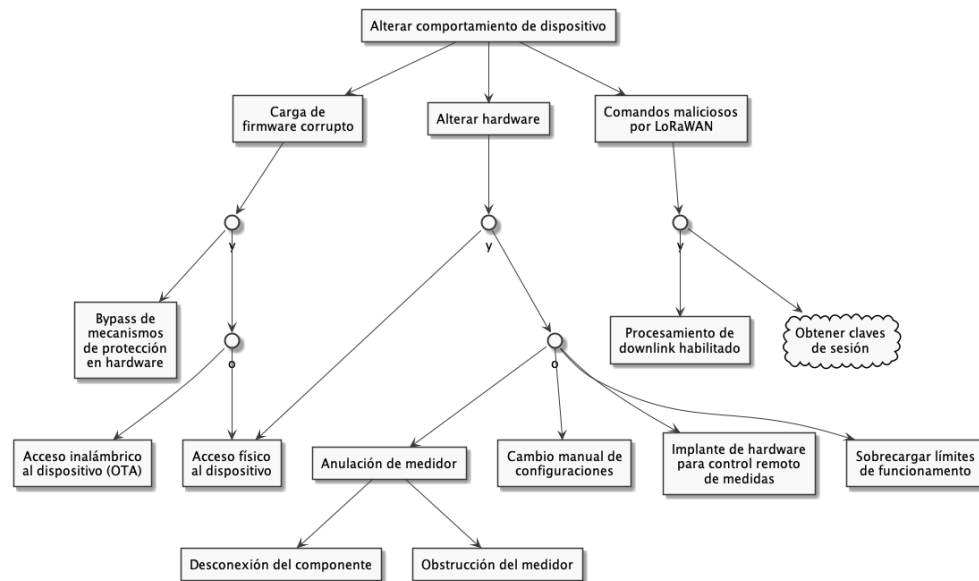


Figura 17: Árbol de ataque para el objetivo “Alterar comportamiento de dispositivo”.

La siguiente rama menos compleja consiste en poder enviar comandos al ED a nivel de aplicación, es decir en el *payload* de mensajes *downlink* de LoRaWAN. Esto claramente es viable si el ED soporta procesamiento de *downlink* y se tienen las claves de sesión para manipular a criterio los comandos soportados. No es el enfoque de este estudio analizar la posibilidad que un ED acepte comandos, pues el análisis se extendería y se cree recomendable realizar un estudio separado con esta consideración desde el inicio. Aquí se agrega por completitud de este objetivo de ataque.

La última rama busca afectar físicamente el *hardware* del ED de distintas formas. Bajo la asunción de que el ED toma medidas, sería posible anularlas de forma externa (es decir, sin desarmarlo) obstruyendo físicamente el componente de *hardware* que oficia de sensor. También si el sensor es independiente del circuito integrado del MCU que procesa las medidas, podría cortarse su conexión, dejando que el ED reporte medidas constantes o inconsistentes. Otra posibilidad de cambiar comportamiento es que el ED cuente con una interfaz (por fuera o por dentro) como botones o teclados para configurar sus parámetros de forma manual. Un ataque más sofisticado consiste en insertar un implante de *hardware* en la conexión entre sensor y MCU, que además de tener la capacidad de enviar al MCU medidas arbitrarias pueda ser controlada remotamente por alguna tecnología inalámbrica. Así un ED podría ser modificado y colocado en su lugar original para luego controlarlo a demanda. Sobre esto, la investigadora Sheila Berta presentó en el evento Black Hat de 2019 cómo inyectar una puerta trasera (*backdoor*) en el *firmware* original de un MCU para poder comunicar datos internos a otro periférico [32]. Por último, se identifica la sobrecarga de límites de funcionamiento, haciendo referencia a sobrecargar un sensor con su respectiva magnitud (por ejemplo con temperaturas muy altas/bajas, sonidos muy fuertes) posiblemente dejando al ED en un estado indefinido.

3.5.2.4 A05 - Modificación indebida de datos

La modificación indebida de datos es apenas distinta a la de mensajes descrita en 3.5.2.1, de hecho, se consideró el objetivo A01 como una de las ramas del objetivo A05. Las otras dos formas de afectar al dato que se identificaron son enviar mensajes antiguos en la misma sesión LoRa-WAN (ataque de *replay*) y nuevamente *bit-flipping*, pero esta vez en el enlace NS-AS (Figura 18).

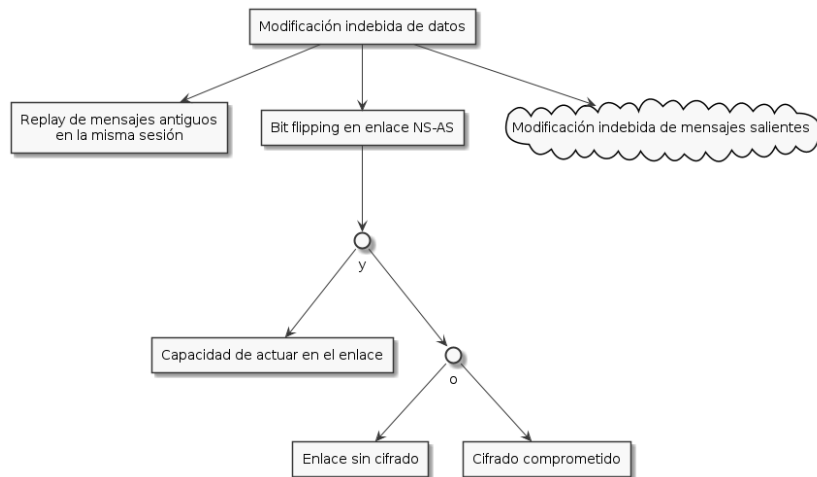


Figura 18: Árbol de ataque para el objetivo “Modificación indebida de datos”.

El *replay* impacta diferente según el tipo de activación de EDs. Si es OTAA, lo usual es que cuando se agote el contador FCnt, en lugar de comenzar de 0 nuevamente, se inicie un *Join Procedure* para renovar las claves de sesión, evitando que se puedan usar mensajes con un valor de FCnt repetido. Cambiando las claves la firma y el cifrado ya serán distintos, aunque se comience de 0

nuevamente. Por lo que para llevar acabo el ataque en OTAA, el ED tendría que hacer reuso del FCnt y seguir en la misma sesión LoRaWAN. De hecho, si la activación es ABP, este problema se da constantemente una vez que se reinicia el FCnt porque las claves no cambian. Aquí entra en juego el largo del FCnt, que en 2.3.3.2 se explica cómo funcionan los contadores en 16 y 32 bits. Haciendo cálculos, suponiendo que un ED envía un mensaje cada un minuto (frecuencia bastante alta considerando que se apunta a bajo consumo) con un FCnt de 16 bits, el contador se agotaría luego de $2^{16} = 65536$ mensajes, es decir en 65536 minutos que traducido a días serían: $65536 \div 60 \div 24 \cong 46$ días (tolerable para un atacante). En cambio, la misma cuenta con 32 bits en años es: $2^{32} \div 60 \div 24 \div 365 \cong 8172$ años. Por supuesto la frecuencia de los mensajes afecta la duración y dependiendo de esta un contador de 16 bits podría llegar a ser viable. Si por motivos técnicos no es posible ampliar el contador en el ED o NS en ABP, es necesario por seguridad cambiar las claves de sesión. Otra forma de hacer efectivo el *replay* es que no se valide el FCnt en el NS, caso en que se puede enviar **cualquier** mensaje anterior y será considerado válido, algo **no** recomendado.

Finalmente, está el camino de *bit-flipping*. Como se explica en 2.3.3.1, el control de integridad en LoRaWAN termina en el NS, por lo que es responsabilidad del administrador del enlace NS-AS que la integridad se mantenga. La forma de llevar a cabo este ataque requiere la capacidad de actuar en el enlace por MitM o comprometiendo alguno de los extremos y además romper el cifrado utilizado, suponiendo que lo hay.

Para aclarar el impacto, las validaciones de los datos podrían ser pobres considerando que un desarrollador vería poco probable la modificación de estos. Este tipo de asunciones podría reflejarse en potenciales inyecciones SQL si los datos son modificados con control total pensando en los peores casos. La recomendación es mínimamente validar los datos según su tipo en los pasos del flujo que corresponda (donde haya peligro de modificación).

3.5.2.5 A06 - Dejar fuera de la comunicación al dispositivo

A06 afecta exclusivamente la disponibilidad y es un árbol extenso (Figura 19) porque hay diversas formas de denegar la comunicación de un ED, afectando al mismo o al canal. Con el fin de reducir la complejidad de explicación del árbol se comenzará por sus laterales.

El subobjetivo “Indisponibilidad por mantenimiento” refiere a ganar acceso administrativo al dispositivo (objetivo explicado en 3.5.2.2) y desconfigurarlo o dejarlo en un estado no responsivo. Otro subobjetivo evidente es evitar que los mensajes sean intercambiados, ya explicado en 3.5.2.1. Luego hay dos grandes ramas que tienen nodos en común, eliminar el suministro energético o abusar del protocolo.

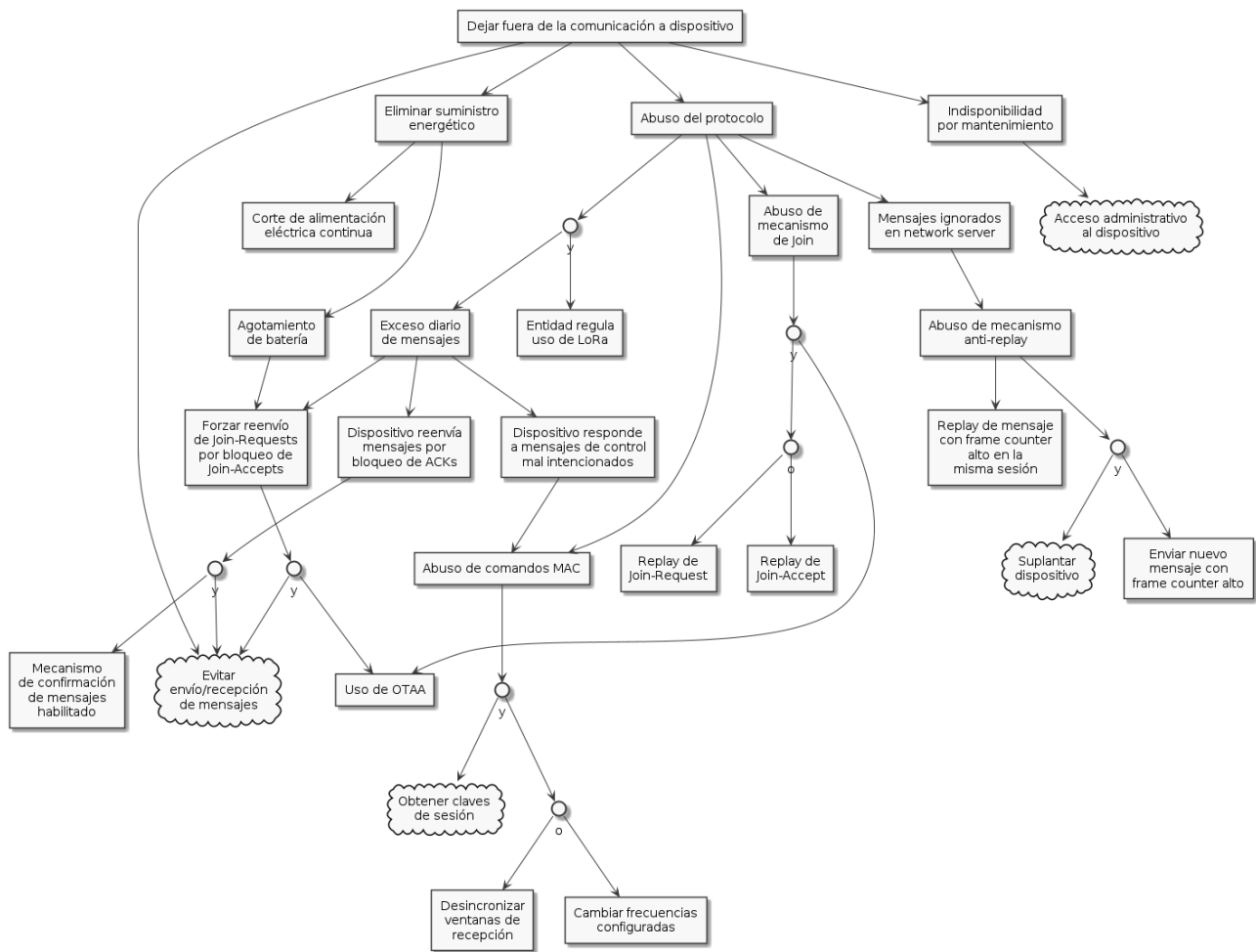


Figura 19: Árbol de ataque para el objetivo “Dejar fuera de la comunicación a dispositivo”.

La energía puede atacarse básicamente cortando el suministro continuo, sin necesidad de acceder físicamente al ED pero sí a la fuente de energía (con acceso físico al ED puede también quitarse la batería si tiene). Este caso es muy básico y es por eso que está concentrado en un nodo. Un ataque más complejo apunta a agotar la batería del ED por otro medio. El identificado es exceder el procesamiento del dispositivo, puntualmente forzando el reenvío de JR (exclusivo para OTAA) optando por bloquear la recepción de los JA. Así evitando que la sesión LoRaWAN nunca se concrete y el ED se vea obligado a reintentar el *Join Procedure* tantas veces como sea necesario hasta iniciar una sesión. Para bloquear los JA se recurre a la rama ya explicada de evitar envío/recepción de mensajes, más compleja en este caso por tener que bloquear mensajes provenientes de potencialmente más de un GW.

El abuso del protocolo puede darse en cuatro subárboles “independientes”. El primero se basa en regulaciones. En 2.3.1 se explica el concepto *duty cycle*, utilizado para evitar el abuso del medio. Las regulaciones no existen en todo el mundo: Uruguay actualmente no tiene regulación por

parte del gobierno en términos de *duty cycle*¹¹, pero Europa sí. El control puede ser intrínseco a la plataforma LoRaWAN, como se da en The Things Network (TTN). TTN es una iniciativa que provee equipamiento e infraestructura para dar soporte a una red LoRaWAN global y abierta, con gestión de datos en la nube. Las redes pueden ser públicas o privadas, en el caso de las públicas TTN aplica una política llamada *Fair Access Policy* que limita al ED el tiempo en aire de *uplink* a 30 segundos por día por sub-banda y para *downlink* a 10 mensajes por día. Hoy en día para Uruguay no es una amenaza, pero podría serlo eventualmente y por eso se incluye en el análisis. El ataque entonces es exceder el envío de mensajes del ED que puede ser por JR extra (ya explicado), reenvíos por falta de confirmación (ACK) y respuestas a mensajes de control del atacante. Si se implementa, los mensajes de *uplink* que necesitan confirmación esperan un correspondiente ACK por *downlink*, y si este se bloquea (por los ataques ya mencionados) el ED intentará al menos una vez más hacer llegar el mensaje al NS (en el peor de los casos envía permanentemente). Los mensajes de control mal intencionados se explican a continuación.

El segundo subárbol relativo al abuso del protocolo y que se une con el anterior es abusar de los comandos MAC de LoRa [21]. Estos mensajes que requieren firmarse con un MIC (por eso es necesario obtener las claves de sesión) tienen el poder de afectar la comunicación por radiofrecuencia. Los ejemplos identificados son enviar comandos para modificar en qué frecuencias emitir (con valores fuera del rango esperado por el GW) y desincronizar las ventanas de recepción, que afecta a los ED que implementan clase B. Si se eligen valores correctos para las ventanas, el NS estará enviando mensajes *downlink* en tiempos incorrectos que nunca serán recibidos por el ED. La naturaleza de estos ataques vulnera el hecho de que un ED no tiene forma de verificar el origen de estos comandos y cree que provienen de un GW legítimo. Si los ataques se dan en competencia con mensajes legítimos, deben enviarse al ED antes o con una señal más fuerte.

El tercer subárbol aplica solo a OTAA y se basa en *replay* de JA o JR [34]. Como se vio al final de 2.3.3.3, el NS almacena N_D DevNonces confirmados de JR previos y revisar esta lista es su forma de protegerse ante ataques de *replay*. Sin embargo, eventualmente un DevNonce quedará fuera de ese listado a medida que se aceptan nuevos, dando la oportunidad de atacar con un JR viejo. Un atacante no puede confirmar una nueva sesión con un JR replicado porque precisa enviar un mensaje *uplink* válido en la nueva sesión, por lo que el ataque explota otro aspecto, la cantidad sesiones sin confirmar N_S en el lapso entre que el ED envía un JR y el NS recibe el mensaje confirmando la nueva sesión. Si se envían $N > N_S$ JRs (con DevNonces no guardados en los N_D mencionados), el NS se vería obligado a descartar la sesión sin confirmar por el ED. Lo que sucede finalmente es que el ED cambia su contexto de sesión asumiendo que el NS también lo hizo, pero en el NS no se terminó de confirmar ninguna sesión por lo que sigue con la anterior y se desincronizan, descartándose todos los nuevos mensajes del ED.

Una posible defensa del NS es bloquear EDs que intentan repetir DevNonces, con la contra importante de que un atacante puede hacer DDoS con JR recolectados de varios EDs. Una variante combinada con forzar el reenvío de JR, es que se reenvíen suficientes JR para repetir

¹¹La Dirección Nacional de Comunicaciones resolvió en el año 2000 una limitación de ocupación continua de canal (*dwell time*) para la banda de frecuencias 902-928 MHz, en la resolución 35 2000, artículo 1, sección 6.2. Este valor se restringe en promedio a máximo 0,4 segundos durante un período de 20 segundos [33].

un DevNonce y responder con un JA viejo, apoderándose de la nueva sesión [34]. DoS por *replay* de JA es más viable porque basta con esperar que el ED envíe un JR con el DevNonce de un JA viejo y enviarle este al ED para que derive las claves de sesión iniciando un nuevo contexto de sesión del que el NS no tendrá noción. Esto es posible porque el ED en LoRaWAN 1.0.3 no valida los cambios de contexto de sesión, es decir, no lleva registro de sesiones anteriores.

El cuarto subárbol es un subobjetivo que en parte ya se da en anteriores. Aquí se busca que mensajes sean ignorados explícitamente por abuso de configuración recomendada del protocolo, particularmente del mecanismo *anti-replay* por control de *frame counter* en el NS. En primer lugar, debe estar habilitado este control (totalmente recomendado con su debido control del límite de FCnt para evitar reinicios). Luego suplantando al ED es posible enviar mensajes con FCnt arbitrariamente alto, tomándose en el NS como válido y posteriormente descartando **todos** los mensajes legítimos del ED con un FCnt menor al del mensaje malicioso. Un ataque más básico se logra quitando el requerimiento de suplantar al ED (no se tienen claves de sesión) y enviando un mensaje viejo con FCnt alto dentro de la misma sesión. Esto funciona si se reinicia el valor del FCnt a 0 y el ED sigue con la sesión (ABP y posiblemente en OTAA), donde en su mitigación hay que considerar lo ya explicado sobre largo de contadores en 3.5.2.4.

Como último detalle relacionado a los ataques que esperan valores específicos de DevNonce, dos investigaciones revelan que el DevNonce podría llegar a ser influenciado afectando el generador pseudoaleatorio [23, 35]. En este ejemplo, el transceptor SX1272 obtiene la secuencia aleatoria basándose en N lecturas del bit menos significativo del indicador de fuerza de señal (RSSI, *received signal strength indicator*) que debería variar constantemente por ruidos y otros factores naturales. Entonces de afectar el RSSI para tener valores constantes podría generarse un DevNonce arbitrario. Es un ataque complejo y dependiente del *hardware*, pero vale la pena mencionarlo.

3.5.3. Análisis con SHARPE

Un árbol de ataque no sirve únicamente para desglosar gráficamente los ataques e identificar contramedidas para estos, sino que también pueden ser analizados desde el punto de vista probabilístico para deducir qué tan probable podría ser el éxito de un ataque y cuáles hojas es conveniente mitigar. Esto último es porque no siempre se tiene los recursos para reducir los riesgos o el diseño no lo permite para todos los nodos.

SHARPE (*Symbolic Hierarchical Automated Reliability and Performance Evaluator*) [36, 37] es una herramienta que como su nombre lo indica, permite evaluar rendimiento y confiabilidad de ciertas soluciones. No se hará listado exhaustivo de sus funcionalidades, pero se destacan las relacionadas a modelos de confiabilidad combinatoria: diagramas de confiabilidad en bloque, grafos de confiabilidad y árboles de falla. Estos últimos son los referentes en el análisis probabilístico a los árboles de ataque. Sus nombres difieren pues el resultado de hacer un árbol de falla muestra cuan confiable y seguro es un sistema, mientras que un árbol de ataque muestra qué tan factible es explotar un sistema a través de algún vector de ataque. En otras palabras, el porcentaje de éxito en un ataque en el contexto de un árbol de ataque se corresponde con el porcentaje de falta de confiabilidad y seguridad cuando se considera un árbol de falla.

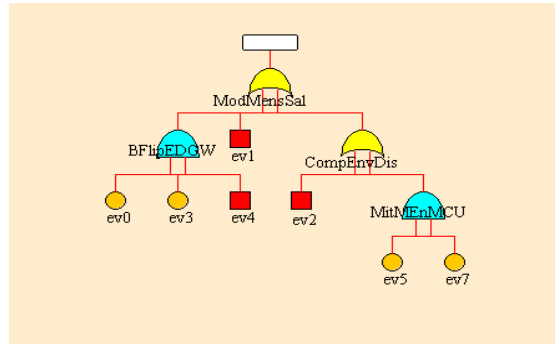
3.5.3.1 Diseño

SHARPE provee una interfaz gráfica donde los árboles pueden ser modelados para luego asignar probabilidades de falla (entre otros tipos de valores) a sus hojas, que se consideran como probabilidades de éxito de las hojas. Dada la complejidad del análisis cambiando tales valores, se decidió acotar las evaluaciones a uno de los árboles más críticos: “Modificación indebida de mensajes salientes” cubierto en 3.5.2.1. El árbol de la Figura 12 y los subárboles correspondientes fueron diagramados en los diferentes árboles de SHARPE de la Figura 20. El resto de los árboles fue también diagramado y los archivos de SHARPE correspondientes serán anexados a este informe para uso futuro.

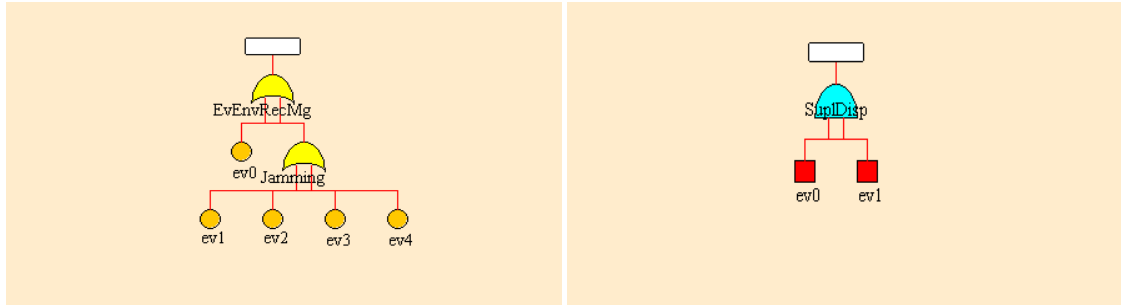
Los nodos con hijos son representados como compuertas AND (celestes) y OR (amarillas) o como nodos (rojos) que simbolizan una “jerarquía” de SHARPE, lo que se traduce a un subárbol. Las hojas son los círculos naranjas y se les asignan nombres automáticamente con la nomenclatura “ ev_i ” siendo i un entero mayor o igual a 0. Los nodos con triángulos representan repetición dentro del mismo árbol de falla, por ejemplo, en la Figura 20.f los nodos ev_{11} y ev_3 son “Filtración de datos en el proveedor” por lo que su valor se repite. De igual manera, los nodos ev_1 y ev_9 son el subárbol “Robar claves de dispositivo” de la Figura 20.g.

Es de esperar que surja la interrogante de por qué un árbol se separó en siete. Por más que podría haberse diagramado un único árbol, tendrían que haberse tenido en cuenta todos los nodos repetidos (como el de acceso físico al dispositivo) y también se reiterarían subárboles, como es el caso de “Robar claves de dispositivo”. Esto ya se dio construyendo el árbol principal en cuestión y claramente surgió con otros de los árboles principales. El reúso en SHARPE resultó vital en términos de practicidad y es una funcionalidad valiosa.

Puede observarse también que la cantidad de nodos en SHARPE en algunos casos es menor a los que se ven en las figuras de la sección 3.5.2. Esto es porque SHARPE no permite cadenas de nodos, siempre debe haber compuertas con al menos dos hijos que pueden ser compuertas u hojas, algo natural para la lógica de un árbol de ataque. Los árboles utilizados en 3.5.2 son representaciones para explicar y se puso foco en que fueran entendibles más que cumplir con una notación estricta de diagramación.

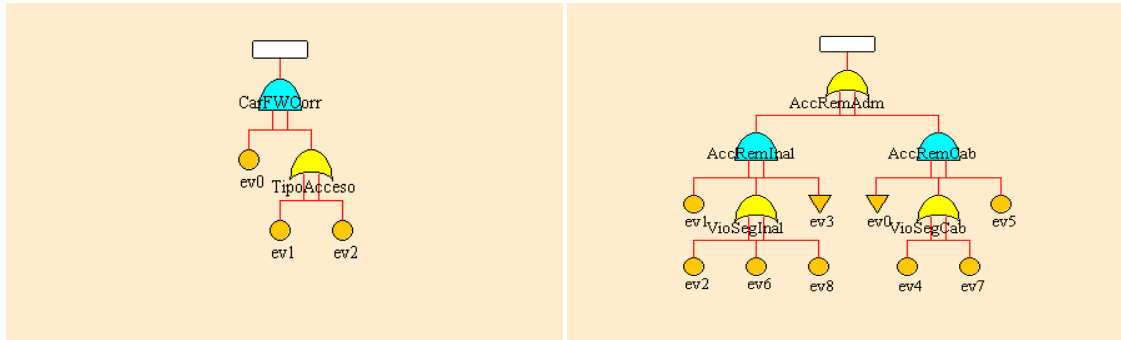


(a) Modificación indebida de mensajes salientes



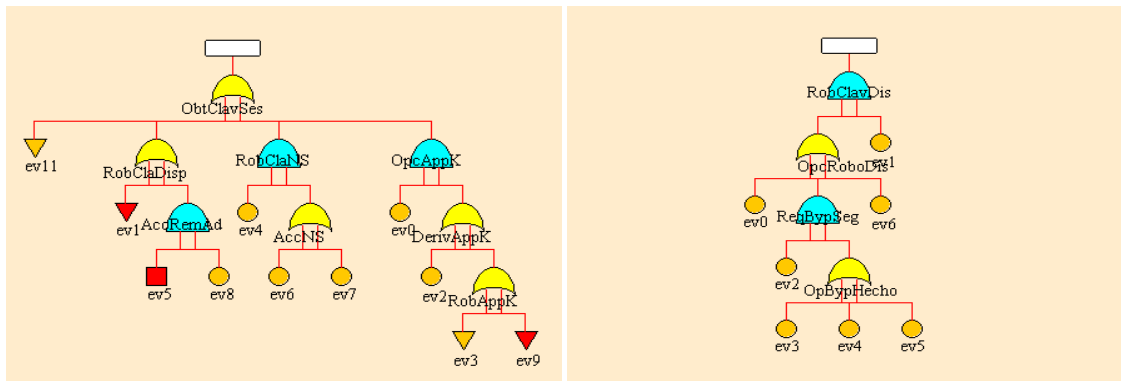
(b) Evitar envío/recepción de mensajes

(c) Suplantación de dispositivo



(d) Carga de *firmware* corrupto

(e) Acceso remoto administrativo al dispositivo



(f) Obtener claves de sesión

(g) Robar claves de dispositivo

Figura 20: Árbol en SHARPE de modificación indebida de mensajes salientes junto a sus subárboles correspondientes. La herramienta no permite nombres medianamente largos en las compuertas.

3.5.3.2 Análisis probabilístico

Con los modelos contruidos en SHARPE el siguiente paso es asignar probabilidades base a las hojas y ejecutar un análisis de confiabilidad de árboles de falla. Lamentablemente en la actualidad no existen datos estadísticos suficientes que permitan definir probabilidades certeras y aunque los hubiera, los números varían según la realidad, como ya se planteó para las prioridades de objetivos de ataque en 3.5.1. Por este motivo se decidió plantear un escenario básico con sensores IoT de una ciudad inteligente con activación ABP. Por básico además se asume que los sensores no cuentan con interfaz administrativa, sino que para mantenimiento deben verse por el proveedor y cargar nuevo *firmware*. Fijado el escenario, se consideran las probabilidades de la Tabla 4.

<i>Hoja</i>	<i>Probabilidad</i>	<i>Observación</i>
Uso de OTAA	0	Se asume activación ABP.
Deducir AppKey	0	Se asume activación ABP.
Interfaz administrativa disponible	0	Se asume sin interfaz administrativa.
Falta de control de MIC	0,01	Los MICs se controlan por especificación. No controlarlos es muy poco probable y viene de una mala implementación.
Claves impresas en dispositivo	0,01	A pesar de que existen casos y por eso se incluye, se considera probabilidad baja.
<i>Exploit</i>	0,02	Se toma de estadística de CERTuy por <i>malware</i> [38].
Conocimiento de formato de <i>payload</i>	0,05	Requiere visibilidad de un <i>payload</i> válido en logs o documentación, tiende a corresponder a un acto interno.
Circuito de integrado LoRa independiente del micro-controlador	0,05	Varía entre fabricantes y no se tiene forma de medirlo. Se asume probabilidad baja contemplando su complejidad.
Reducción de rango por bloqueo físico de señal	0,1	No se investigó la influencia de materiales que obstruyan la señal en LoRa. Contemplando que <i>jamming</i> es una técnica efectiva y funcional, es preferible recurrir a ella que a bloqueo físico. Se considera probabilidad baja.
Filtración de datos en el proveedor	0,12	Según el último informe de Verizon [39], de 32002 incidentes que cumplían sus estándares de calidad, 3950 fueron filtraciones de datos comprobadas. Se utilizan estas cifras para un estimado con estadística, aunque no sea local.
<i>Bypass</i> de mecanismos de protección en <i>hardware</i>	0,2	Considerando los resultados de experimentación (ver documento anexo “ <i>Anexo - Experimentación en LoRaWAN</i> ”), se asume probabilidad media-baja. Aunque se cree que con <i>expertise</i> previo podría lograrse.
Claves se filtran en mensajes de log	0,2	Es muy común que existan mensajes de <i>debug</i> en log por alguna interfaz. Experimentando no se vieron claves, pero por errores de configuración, descuido o pereza podrían filtrarse. Se considera probabilidad media-baja.

Continuación de la Tabla 4 de la página anterior

Claves se almacenan en archivos de texto	0,2	Aunque es más fácil de conseguir y leer un archivo de texto, es más probable que las claves estén cargadas en registros de memoria no volátil. Se considera probabilidad media-baja.
Análisis de memoria del dispositivo	0,3	Considerando los resultados de experimentación y ejemplos referenciados, se asume probabilidad media. Aunque se cree que con <i>expertise</i> previo podría lograrse.
Ingeniería reversa en dispositivo	0,3	Considerando los resultados de experimentación y ejemplos referenciados, se asume probabilidad media. Aunque se cree que con <i>expertise</i> previo podría lograrse.
Acceso a red interna	0,3	Se toma de estadística de CERTuy por compromiso de sistemas [38].
Obtener credenciales	0,37	Hay muchas formas de obtener credenciales (fugas de datos, <i>phishing</i> , ingeniería social, <i>malware</i> , errores humanos, entre otras), sin embargo, se toma de estadística de CERTuy por <i>spam/phishing</i> que es de los motivos más populares [38].
Acceso inalámbrico al dispositivo (OTA)	0,5	Depende del fabricante o proveedor tenerlo habilitado por practicidad, se considera probabilidad media.
<i>Jamming</i> continuo	0,7	Es totalmente funcional, aunque se baja su probabilidad por potencialmente levantar sospechas.
Acceso físico al dispositivo	0,8	En un escenario de ciudad inteligente es probable que los dispositivos se encuentren en la vía pública y protegerlos de cualquier acto de acceso físico es complejo.
<i>Jamming</i> reactivo	0,8	Ídem a <i>jamming</i> continuo porque no sigue criterios, aunque no está activo al 100 % y esto puede favorecer a evitar su detección.
<i>Jamming</i> selectivo + <i>wormhole</i>	0,85	Probabilidad alta respecto a continuo y reactivo debido a su criterio de selección. Baja levemente su probabilidad por la complejidad del ataque.
<i>Jamming</i> selectivo	0,9	Probabilidad alta respecto a continuo y reactivo debido a su criterio de selección.

Tabla 4: Asignación de probabilidades en SHARPE para el caso particular planteado.

Puede notarse que no se han incluido probabilidades para los nodos del subárbol “Acceso remoto administrativo al dispositivo”. Esto es porque al asumir que no se tiene interfaz administrativa disponible, al estar bajo una compuerta AND con una hoja de probabilidad 0, la probabilidad de falla del subárbol completo se desprecia.

SHARPE dispone de un analizador con múltiples funciones que toman como entrada los modelos con sus valores (en este caso árboles de falla con sus probabilidades de falla) y retornan el resultado de la ejecución de las funciones seleccionadas. El análisis con probabilidades de falla que sirve para este caso se basa en la función *Unreliability*, la cual puede invocarse con un valor

fijo de tiempo t o en iteraciones con tiempo inicial, tiempo final y paso (incremento por iteración). SHARPE genera un código de análisis que define las estructuras y probabilidades con una sintaxis particular, que puede ser editado. A continuación, se muestra el código parcial¹² para el caso planteado, con la definición del árbol de falla principal dado que es el ejemplo más completo y de la ejecución de *Unreliability*:

```

1  ...
2  ftree ModificacionIndebidaDeMensajesSalientes
3  basic SuplDisp cdf(SuplantacionDeDispositivo)
4  basic EvEnvRecMens cdf(EvitarEnvioRepcionDeMensaje)
5  basic CargFWCorr cdf(CargaDeFirmwareCorrupto)
6  basic ConFrmtPayl prob(0.05)
7  basic FalContMIC prob(0.01)
8  basic CircLoRaIndMCU prob(0.05)
9  basic AccFisDisp prob(0.8)
10 and MitMenMCU AccFisDisp CircLoRaIndMCU
11 or CompEnvDis CargFWCorr MitMenMCU
12 and BFlipEDGW FalContMIC ConFrmtPayl EvEnvRecMens
13 or ModMensSal BFlipEDGW SuplDisp CompEnvDis
14 end
15
16 echo *****
17 echo ***** Outputs asked for the model: ModificacionIndebidaDeMensajesSalientes *****
18
19 var Unreliability tvalue(1;ModificacionIndebidaDeMensajesSalientes)
20 expr Unreliability
21
22 end

```

Las líneas previas a este código abarcan la definición de árboles, cubriendo las dependencias para árboles que se definen después. Es decir, el árbol de falla **ModificacionIndebidaDeMensajesSalientes** definido con la expresión **ftree** utiliza árboles definidos previamente con la misma expresión. Los nodos (líneas 3 a 9) son definidos con la palabra **basic**¹³, el nombre de nodo y luego su valor. Los nodos con probabilidad llevan **prob(x)** siendo **x** su valor y, si son subárboles, llevan **cdf(N)** siendo **N** un nombre definido previamente con **ftree**. Una vez definidos los nodos hoja, se construyen las ramas con **and** y **or**, agregando el nombre de la compuerta y luego los dos nodos/compuertas que conecta. La definición del árbol se finaliza con **end**.

Con todos los árboles de falla definidos y conectados se procede a evaluar la falta de confiabilidad. Con **var** se define una variable llamada **Unreliability** que contiene la expresión de evaluación del árbol principal en el tiempo $t = 1$, escrito como:

tvalue(1;ModificacionIndebidaDeMensajesSalientes)

Luego se evalúa efectivamente la expresión de la variable con **expr**. Dos detalles sobre la evaluación son que el uso de **var** no es realmente necesario (podría usarse **expr** con la llamada a **tvalue**) y que existe también una función *Reliability*, pero es simplemente $1 - \text{Unreliability}$, la probabilidad complementaria.

¹²El código completo no es relevante para este análisis. Se basa en todos los modelos de la Figura 20.

¹³Si es un nodo que se repite en el mismo árbol, se define con **repeat**.

Finalmente, el código se ejecuta y para las probabilidades base de la Tabla 4 se imprime el siguiente resultado:

```

1 *****
2 ***** Outputs asked for the model: ModificacionIndebidaDeMensajesSalientes *****
3 Unreliability: 5.36033031e-001
4
5 -----

```

Esto se traduce a que la falta de confiabilidad es aproximadamente del 53,6 % que interpretado desde la perspectiva de árbol de ataque, es la probabilidad de cumplir con el objetivo raíz a través de algún vector de ataque. Es un valor relativamente alto tratándose de una amenaza crítica. Ahora el objetivo es observar cómo impacta la reducción de probabilidades en consecuencia de implementar algunas contramedidas de forma acumulativa.

Se comienza con un caso de resultado obvio, eliminando la opción de acceso inalámbrico al dispositivo, ya sea por deshabilitarla o que este carezca de ella. Esto reduciría la probabilidad de 0,5 a 0, pero es una hoja de la que depende solo un camino hacia la raíz y es poco influyente. De hecho, el resultado con esta contramedida muestra una reducción con diferencia de $\sim 1,2\%$:

Unreliability: 5.24716764e-001

Se propone entonces mitigar el *bypass* de mecanismos de protección en *hardware* que cuenta con una probabilidad mucho menor que el caso anterior (0,2 contra 0,5), pero es incluido en más de un subárbol y es condición necesaria para llevar a cabo varias ramas (está bajo compuertas AND). No se asumirá reducción a 0, pero investigando que no existan vulnerabilidades conocidas en el *hardware*, se cuente con mecanismos como cifrado de *flash* y *secure boot*, y se utilice un *hardware security module* para la gestión de datos sensibles y operaciones criptográficas, se estima que la probabilidad podría bajar considerablemente. Estimando una reducción de 0,2 a 0,01 el nuevo resultado es:

Unreliability: 3.84485554e-001

Esta reducción con diferencia de $\sim 14\%$ prueba la importancia de enfocarse en hojas influyentes, aunque tengan menor probabilidad que otras.

El acceso físico es el siguiente nivel, es decir, tiene una alta probabilidad y además es dependido por varios vectores de ataque. Es una de las hojas más interesantes y para evidenciarlo se asumirá una reducción de su probabilidad de 0,8 a 0,05. Contramedidas posibles serían alejar los EDs del alcance público sin afectar el rendimiento (ubicarlos en alturas significativas, por ejemplo), añadir vigilancia por cámaras y protecciones físicas alrededor del ED¹⁴. El nuevo resultado con contramedidas es:

Unreliability: 2.31805446e-001

Se evidencia una reducción con diferencia de $\sim 15\%$ que apoya su relevancia en el árbol. Es importante comentar que las hojas que no superen en probabilidad a las de sus hojas hermanas (del mismo nivel) o resultados de subárboles hermanos, no tendrán influencia respecto al resto si se

¹⁴Obviamente no siempre es viable física o económicamente aplicar todas estas medidas, es un caso hipotético para justificar la reducción.

encuentran bajo una compuerta OR pues se toma el resultado parcial o probabilidad mayor. Por ejemplo, La reducción de probabilidad de acceso físico implica también bajar la probabilidad de la reducción de rango por bloqueo físico de señal, sin embargo, esta hoja es hermana del subárbol de *jamming* bajo una compuerta OR. Por lo tanto, su probabilidad es opacada y el resultado es igual que si no hubiese bajado.

Otra forma de analizar la importancia de una hoja es observando la variación de su probabilidad en un rango de valores y distinguir si existen diferencias significativas. Ejemplificando nuevamente con el acceso físico, se realiza el análisis con valores de 0 a 1 variando de a 0,25. En la Tabla 5 se muestra la probabilidad de falla total para cada caso.

<i>Probabilidad de hoja</i>	<i>Probabilidad de falla total</i>
0	22,12 %
0,25	27,36 %
0,5	32,48 %
0,75	37,47 %
1	42,33 %

Tabla 5: Variaciones de probabilidad para hoja de acceso físico al dispositivo y sus impactos en la probabilidad de falla total.

Los resultados son congruentes con la relevancia de la hoja, implicando una variación máxima de $\sim 20,21$ % con la presencia o ausencia de la amenaza. Incluso, esta diferencia es para el resultado con contramedidas acumuladas, pero partiendo de las probabilidades base y contando cambios únicamente para esta hoja, la probabilidad de falla total es $\sim 29,91$ % para 0 y $\sim 58,8$ % para 1, marcando una variación máxima de $\sim 28,89$ %.

Se concluye este análisis resaltando la utilidad de SHARPE y sus evaluaciones en árboles de fallas. También recomendando identificar los nodos con más dependencias independientemente de sus probabilidades de falla, especialmente si estas dependencias se dan bajo compuertas AND. El análisis podría ser extendido al resto de las hojas y árboles principales, pero como se explica en 3.5.3.1, se tornaría complejo y extensivo, el punto aquí era presentar SHARPE y sus beneficios respecto al análisis probabilístico de árboles de ataque.

4. Experimentación

En esta sección se detallan las pruebas ejecutadas (con o sin éxito) sobre los dispositivos con los que se contó en el desarrollo del proyecto y que siguen la línea de ataques viables identificados en 3.5.2, con la salvedad de pruebas que contienen datos potencialmente sensibles para la Intendencia de Montevideo (para ver la experimentación completa, referirse al documento “*Anexo - Experimentación en LoRaWAN*” separado de este informe¹⁵). No todas las pruebas pudieron realizarse por cuestiones de experiencia, equipamiento o tiempo. En cuanto a los dispositivos, por colaboración del grupo MINA de Facultad de Ingeniería se obtuvo la mayoría de estos, contando además con un sonómetro provisto por la Intendencia de Montevideo y algún material adicional obtenido por medios propios. El listado y detalles de los dispositivos se encuentra en la siguiente sección.

4.1. Dispositivos utilizados

4.1.1. SparkX ESP32 LoRa 1-Channel Gateway

SparkX ESP32 LoRa 1-Channel Gateway [13] está compuesto por un microcontrolador programable ESP32 [12] y un transceptor LoRa RFM96W [40]. La versatilidad de ser programable permitió usar dos de estas placas para realizar diferentes funciones. Inicialmente, una de ellas se configuró como un dispositivo IoT, usando el *firmware* Lua RTOS [41]. La otra placa se configuró como un GW [42], haciendo uso del *firmware* ESP-1ch-Gateway [43]. En el desarrollo del proyecto se fueron cargando los *firmwares* que figuran en la experimentación.

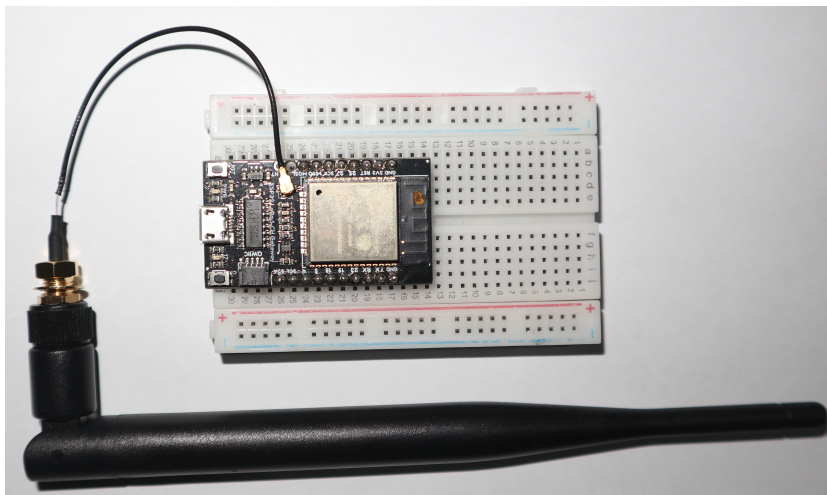


Figura 21: SparkX ESP32 LoRa 1-Channel Gateway sobre *protoboard* con antena conectada.

4.1.2. The Things Gateway

The Things Gateway [44] de The Things Network (abreviado TTN en adelante) es un GW LoRa de fácil configuración que se conecta mediante Wi-Fi a Internet. La gestión del GW y visualiza-

¹⁵El documento es de uso exclusivo de la IM y de quienes les sea concedido su acceso por ella.

ción los mensajes recibidos se hace mediante una consola web provista por TTN [45].

La activación y configuración del GW se hace realizando un reinicio de este, en caso de no estar en su estado inicial, presionando rápidamente dos veces seguidas el botón “mode” y esperando que los cuatro LEDs parpadeen. Luego de esto se debe acceder a un sitio de activación de TTN [46] y seguir las instrucciones.



Figura 22: The Things Gateway con antena conectada.



Figura 23: The Things Gateway con antena conectada y sin tapa superior.

4.1.3. The Things Node

El dispositivo The Things Node [47] es programable mediante Arduino y está basado en un SparkFun Pro Micro - 3.3V/8MHz que adicionalmente tiene un módulo LoRaWAN, un sensor de temperatura, un acelerómetro digital, un sensor de luz, un botón y un LED RGB. Además,

puede ser alimentado por un cable USB o con tres baterías AAA.

Para poder programarlo es necesario configurar el ambiente previamente según la documentación de TTN [48], esta indica que es necesario instalar las librerías **TheThingsNetwork** y **TheThingsNode** mediante el gestor de librerías de Arduino y para soportar su placa se debe también agregar las placas SparkFun mediante las instrucciones de esta guía [49].



Figura 24: The Things Node, vista frontal.



Figura 25: The Things Node, vista trasera sin tapa.

4.2. Captura de mensajes emitidos por el sonómetro

Mediante el uso del GW de TTN (The Things Gateway) se realizó la captura de los mensajes enviados por el sonómetro. Para ello, se configuró el GW como se detalla en la sección 4.1.2, con la particularidad de que inicialmente se había configurado con el plan de frecuencia de USA, pero luego se detectó que debía de ser el de Australia, debido a las frecuencias que el sonómetro está utilizando. Si bien ambos planes figuran identificados como 915 MHz, no usan los mismos canales: US902-928, banda que también esta asignada a Uruguay [50], para el *uplink* usa el rango 903.9-905.3 [51], mientras que AU915-928 usa el rango 916.8-918.2 **en TTN**. Ambas bandas

comparten el mismo rango para *downlink*, 923.3-927.5.

Al cambiar de plan en la configuración del GW, se observó que es necesario realizar un *reset* completo del dispositivo para que el cambio tome efecto y no funciona si solo se modifica la configuración del GW posterior a haber sido activado. Para poder visualizar los mensajes recibidos por el GW, es necesario acceder al sitio web de TTN¹⁶ (Figura 26). A modo de ejemplo se muestra a continuación la información recibida para uno de los mensajes:

```
{
  "gw_id": "ttn-gw-iot-2",
  "payload": "QGKkgQAYCIBdoCgoILPP8bqswcd",
  "f_cnt": 8904,
  "lora": {
    "spreading_factor": 9,
    "bandwidth": 125,
    "air_time": 185344000
  },
  "coding_rate": "4/5",
  "timestamp": "2020-05-06T02:31:06.048Z",
  "rssi": -73,
  "snr": -10.25,
  "dev_addr": "04928A61",
  "frequency": 917000000
}
```

Se aclara que, en principio no es posible decodificar los datos cifrados del *payload*, sin tener acceso a la AppSKey.

¹⁶<https://console.thethingsnetwork.org>

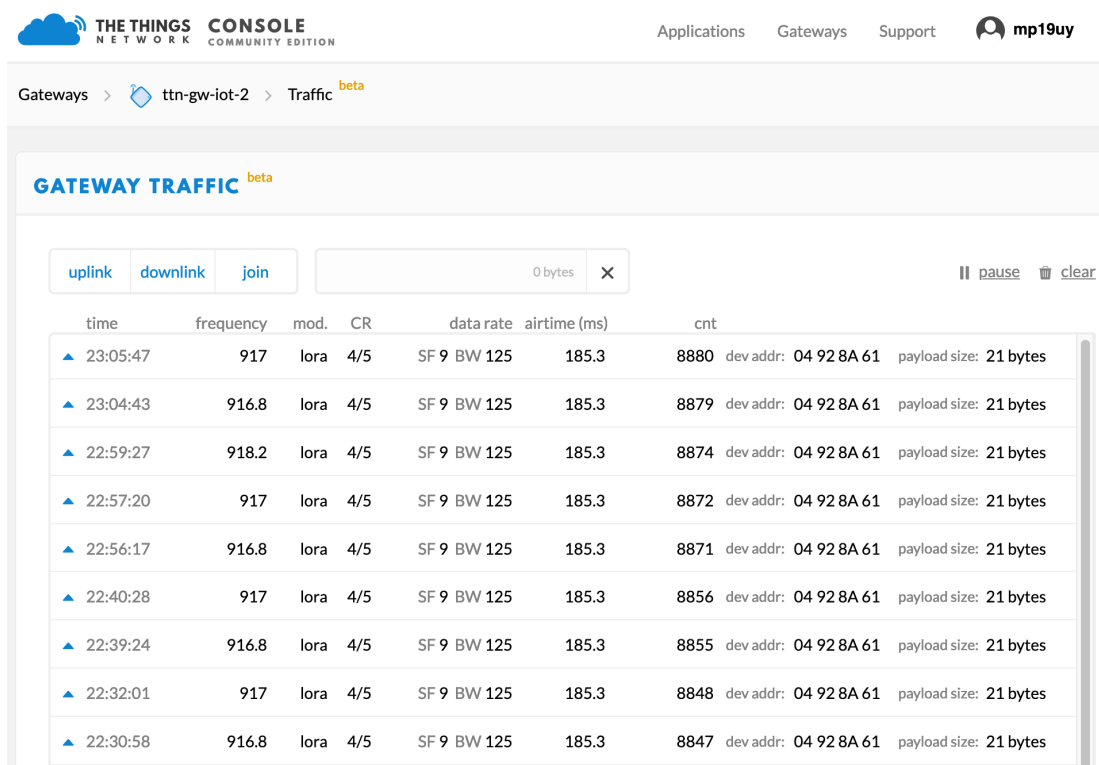


Figura 26: Consola del gateway de The Things Network.

4.3. Herramientas de LoRaWAN Auditing Framework (LAF)

Investigadores de IOActive, a falta de un *framework* para probar y auditar LoRaWAN, desarrollaron uno propio y está público en GitHub [52]. El proyecto busca proveer una serie de herramientas para el armado, *parsing*, envío, análisis y *cracking* de paquetes LoRaWAN, con el fin de realizar auditorías o pruebas de penetración de una infraestructura LoRaWAN.

Aunque algunas funcionalidades están pendientes, las herramientas actuales logran cumplir los propósitos que buscan. Dentro de las herramientas se destacan la recolección de paquetes integrándose a plataformas IoT, el análisis de paquetes recolectados con alertas sobre anomalías y descifrar/cifrar paquetes. Las herramientas de envío fueron hechas para opciones de *hardware* con las que no se contó en este proyecto, por lo que no fueron aprovechadas.

4.3.1. Persistencia de datos recibidos en The Things Gateway

Un problema que presenta el uso de la plataforma de The Things Network, es que en principio no existe un mecanismo que permita la persistencia de los paquetes recibidos por el GW, si estos no están asociados a una aplicación. En el caso de los dispositivos que no se tiene control administrativo ni de su configuración, como lo son los sonómetros (y cualquier otro dispositivo IoT usando LoRaWAN que esté usando el mismo espectro), no se pueden asociar a una aplicación. Si bien los paquetes se pueden visualizar en la interfaz web de TTN como se muestra en la Figura 26, no existe persistencia y solo se mantienen en el navegador web mientras se esté

conectado a esta consola. Como solución a este problema, el *framework* LAF provee un *script* para persistir estos mensajes en una base de datos, para poder analizarlos posteriormente. El *script* se encuentra en la ruta: `auditing/datacollectors/TTNCollector.py` del repositorio [52].

Para poder mantener la ejecución del *script* de manera constante y guardar los mensajes de TTN, se levantó en un proveedor de *cloud* una máquina con Ubuntu 18.04.3 LTS y se instaló Docker y Docker Compose [53], para hacer uso del archivo `docker-compose.yml` que permite levantar la herramienta LAF (junto con la base de datos que requiere) de manera simple y rápida.

Para hacer *build* de la imagen y levantar la herramienta se utilizó el siguiente comando (dentro de la carpeta **laf** del repositorio): `docker-compose up --build -d` y luego se accede mediante `bash` con el comando: `docker exec -ti laf_tools_1 /bin/bash`.

Una vez se tiene acceso a LAF, el comando para iniciar el *script* que guarda los mensajes del GW en TTN es:

```
python3 auditing/datacollectors/TTNCollector.py --user <USUARIO> --pwd <PASSWORD>
--gw-id <ID>
```

Donde <USUARIO> y <PASSWORD> son las credenciales de TTN y <ID> es el identificador del GW.

Para poder dejar ejecutando este *script* de manera constante sin que se detenga al cerrar la sesión SSH, se puede usar la utilidad `screen` de Linux. El *script* guarda los datos en una base de datos PostgreSQL accesible en el puerto local 5432 y además en `docker-compose.yml` se define una instancia de la utilidad `pgAdmin` accesible en el puerto local 5001, para administrar y consultar la base de datos.

loraguard_db/postgres@db

Query Editor

Query History

Scratch Pad

1

2

SELECT id,size,date,gateway,tmst,freq,data,m_type,mic,dev_addr,f_count FROM public.packet WHERE dev_addr = '04928a61' ORDER BY

Data Output

Explain

Messages

Notifications

	id	size	date	gateway	tmst	freq	data	m_type	mic	dev_addr
	[PK] bigint	integer	timestamp with time zone	character varying (30)	bigint	double precision	character varying (300)	character varying (20)	character varying (8)	character varying (8)
1	929	[null]	2020-04-30 07:00:41.284564+00	ttn-gw-iot-2	1588230041236	918200000	QGGKkgQAYQMBb3LHCKR07...	UnconfirmedDataUp	e972c1e1	04928a61
2	928	[null]	2020-04-30 06:52:14.945708+00	ttn-gw-iot-2	1588229534898	918200000	QGGKkgQAwQMBKwYfjZs8Z...	UnconfirmedDataUp	5c06015b	04928a61
3	926	[null]	2020-04-30 06:26:55.854057+00	ttn-gw-iot-2	1588228015808	918200000	QGGKkgQAqQMBHyXITzL8RD...	UnconfirmedDataUp	7ad42b8e	04928a61
4	925	[null]	2020-04-30 06:18:29.681109+00	ttn-gw-iot-2	1588227509635	918200000	QGGKkgQAoQMBwgvzblXW...	UnconfirmedDataUp	49021193	04928a61
5	924	[null]	2020-04-30 05:53:10.660909+00	ttn-gw-iot-2	1588225990615	918200000	QGGKkgQAIQMBZHzEpUwU8...	UnconfirmedDataUp	4ed5d088	04928a61
6	923	[null]	2020-04-30 05:36:17.881091+00	ttn-gw-iot-2	1588224977834	918200000	QGGKkgQAeQMBPxUCI+wfgu...	UnconfirmedDataUp	0b5a905d	04928a61
7	921	[null]	2020-04-30 05:02:32.420166+00	ttn-gw-iot-2	1588222952374	918200000	QGGKkgQAWQMBJWdXYTsKR...	UnconfirmedDataUp	fc0963cb	04928a61
8	919	[null]	2020-04-30 04:54:06.182092+00	ttn-gw-iot-2	1588222446136	918200000	QGGKkgQAUQMBZ7utvQqfGv...	UnconfirmedDataUp	f1ca71b2	04928a61
9	918	[null]	2020-04-30 04:28:47.073912+00	ttn-gw-iot-2	1588220927028	918200000	QGGKkgQAQQMBPHhp/UKSA...	UnconfirmedDataUp	e1ed802d	04928a61
10	916	[null]	2020-04-30 04:18:14.057214+00	ttn-gw-iot-2	1588220294009	916800000	QGGKkgQALwMBnB/HMfgZ2...	UnconfirmedDataUp	b13a583d	04928a61
11	915	[null]	2020-04-30 04:11:54.33226+00	ttn-gw-iot-2	1588219914285	918200000	QGGKkgQAKQMBexWzG1EJO...	UnconfirmedDataUp	5ecf637d	04928a61
12	914	[null]	2020-04-30 04:03:27.978498+00	ttn-gw-iot-2	1588219407928	918200000	QGGKkgQAIQMBHkKkd7qg9j...	UnconfirmedDataUp	d0ebb99f	04928a61
13	913	[null]	2020-04-30 03:55:01.8239+00	ttn-gw-iot-2	1588218901777	918200000	QGGKkgQAGQMBKZw+56do...	UnconfirmedDataUp	7e6e6df	04928a61
14	912	[null]	2020-04-30 03:46:35.362997+00	ttn-gw-iot-2	1588218395316	918200000	QGGKkgQAEQMBISHAEIwOAS...	UnconfirmedDataUp	d347bdf5	04928a61
15	910	[null]	2020-04-30 03:29:42.598914+00	ttn-gw-iot-2	1588217382552	918200000	QGGKkgQAAQMBPdcjX0cW...	UnconfirmedDataUp	eea162a7	04928a61

Figura 27: Mensajes de TTN en base de datos.

4.4. Jamming

En la sección de árboles de ataque (particularmente en 3.5.2.1) se explican los tipos de *jamming* que investigadores describieron en *papers* específicos sobre el tema. El objetivo en este marco fue avanzar cuanto se pudiera en pruebas de concepto que verifiquen la viabilidad e impacto de los tipos de *jamming*. A continuación, se explican todas las pruebas relacionadas, la selección de equipamiento y los *frameworks* de desarrollo.

4.4.1. Equipamiento y librerías

Todo el equipamiento listado en el Anexo ?? fue utilizado en las pruebas. El código de distintas implementaciones de *jamming* fue cargado en las placas SparkX para cumplir el rol de EDs. El dispositivo The Things Node (abreviado TTNode de ahora en más) fue considerado el “ED legítimo” y el GW The Things Gateway (abreviado TTGW) fue una de las piezas fundamentales para detectar las pérdidas de mensajes y ver sus campos no cifrados.

Buscando implementaciones de transmisores LoRa que pudieran cargarse en un SparkX, se encontraron dos opciones viables: una implementación en ESP32 con el *framework* MicroPython [54] y el ejemplo de convertir el SparkX en ED en la guía de GW de un canal de SparkFun [42] (la guía indica cómo instalar las librerías necesarias, se omite su explicación aquí). Dentro de las grandes ventajas de MicroPython se encuentran la versatilidad y poder de utilizar Python, código menos complejo de escribir y entender, y evitar la compilación de librerías. Lamentablemente, la documentación sobre su uso se consideró escasa y logró cargarse en SparkX luego de ya tener ejemplos avanzados siguiendo el otro camino. Aun así se consideró una alternativa interesante para desarrollar en trabajos futuros. La segunda opción fue el paso inicial que llevó a investigar la librería `arduino-lmic` [55] y sus ejemplos. El IDE de Arduino fue la herramienta de carga de código y visualización en la interfaz serial con los dispositivos.

4.4.2. Jamming continuo

El caso básico es la transmisión constante. Como primer paso se tomó el ejemplo de la guía de SparkFun para garantizar que era posible transmitir mensajes LoRaWAN válidos en modo ABP, con éxito. Así se confirmó la viabilidad de utilizar `arduino-lmic`. Sin embargo, se buscó seguir un enfoque de transmisión que evitara el gasto de tiempo que implica generar paquetes válidos LoRaWAN. Revisando los ejemplos de `arduino-lmic` (la versión elegida fue 3.2.0¹⁷) dentro de su mismo repositorio se encontró uno llamado `raw.ino`¹⁸ que como su nombre lo sugiere, envía datos en crudo con LoRa, esto significa que los mensajes enviados **no son LoRaWAN**, no incluyen DevAddr, tipo de mensaje, contadores, ni nada relacionado a la especificación más allá de lo involucrado a radiofrecuencia.

En primera instancia el código fue modificado para únicamente activarse en modo TX (transmisión), recortando todas las funciones relacionadas a RX (recepción) y código irrelevante para las

¹⁷La última versión de `arduino-lmic` liberada al escribir este informe.

¹⁸Disponible en: <https://github.com/mcci-catena/arduino-lmic/blob/v3.2.0/examples/raw/raw.ino>, último acceso: 10 de junio de 2020.

pruebas, por ejemplo, configuración de frecuencias fuera de las bandas US y AU. Se irá explicando el código de a fragmentos omitiendo comentarios y líneas que no aportan a comprender la lógica. Para comenzar, todas las implementaciones con `arduino-lmic` comparten una variable global que define el *pinmap* del *hardware* específico. Esto se mantuvo incambiado del ejemplo de SparkFun:

```
1 // Pin mapping for the SparkX ESP32 LoRa 1-CH Gateway
2 const lmic_pinmap lmic_pins = {
3     .nss = 16,
4     .rxtx = LMIC_UNUSED_PIN,
5     .rst = 5,
6     .dio = {26, 33, 32},
7 };
```

El punto de entrada de los *sketch* cargados con Arduino IDE es la función `setup()`. En ella se escriben todas las sentencias de inicialización necesarias antes de comenzar la ejecución de la función de iteración continua `loop()`. El código relevante de `setup()` y `loop()` es el siguiente:

```
1 void setup() {
2     ...
3     // Initialize runtime env
4     os_init();
5
6     uint8_t uplinkChannel = 8;
7     setupLMICChannel(uplinkChannel);
8     ...
9     os_setCallback(&osJob, tx_func);
10 }
11
12 void loop() {
13     // Execute scheduled jobs and events
14     os_runloop_once();
15 }
```

`arduino-lmic` indica en su documentación del repositorio¹⁹ que se debe llamar a `os_init()` para inicializar el sistema operativo, luego debe encolarse una tarea inicial para ejecutar (`arduino-lmic` trabaja con colas de tareas o *jobs*) y periódicamente se debe ejecutar `os_runloop_once()` para desencolar y correr tareas en espera. La función `os_setCallback()` encola una tarea, en este caso será una llamada a `tx_func()`, la función que iniciará la transmisión continua. En la línea 7 se invoca a la configuración del canal de *uplink* a utilizar, código reutilizado, reducido y encapsulado de `raw.ino`. Aquí se indica que el canal será 8, correspondiente a la frecuencia 903.9 en US y 916.8 en AU (no se implementó salto automático de canales, pero sería posible). Valores como ancho de banda y *spreading factor* (**SF**) se configuran en base al canal seleccionado, aunque no es posible seleccionar (en esta implementación) valores en un rango, por ejemplo, un valor de

¹⁹Disponible en: <https://github.com/mcci-catena/arduino-lmic/blob/v3.2.0/doc/LMIC-v3.0.99.pdf>, último acceso: 10 de junio de 2020.

SF distinto a 7 cuando el canal seleccionado es menor a 64. Para cambiar este tipo de valores es necesario hacerlo dentro de la función `setupLMICChannel()`.

Una vez ejecutada `os_runloop_once()`, se invocarán las funciones asociadas a TX, comenzando por `tx_func()`:

```
1 // Transmit the given string and call the given function afterwards
2 void tx(const char *str, osjobcb_t func) {
3     os_radio(RADIO_RST); // Stop RX first
4     // Wait a bit, without this os_radio below asserts,
5     // apparently because the state hasn't changed yet
6     delay(1);
7     LMIC.dataLen = 0;
8     while (*str) LMIC.frame[LMIC.dataLen++] = *str++;
9     LMIC.osjob.func = func;
10    os_radio(RADIO_TX);
11 }
12
13 static void txdone_func (osjob_t* job) { }
14
15 static void tx_func (osjob_t* job) {
16     tx("Jamming", txdone_func);
17     ...
18     os_setTimedCallback(job, os_getTime() + ms2osticks(CALLBACK_DELAY), tx_func);
19 }
```

Los dos pasos en `tx_func()` son invocar el inicio de transmisión del texto “Jamming” indicando la *callback* (función a ejecutar cuando termine la transmisión, en este caso es vacía) y encolar una tarea con demora llamando a `os_setTimedCallback()`. En el segundo paso la tarea es la misma `tx_func()` para crear un bucle de transmisión y la demora se obtiene consultando el tiempo actual del sistema (`os_getTime()`) y sumándole el tiempo de demora (`CALLBACK_DELAY`) en milisegundos traducido a *ticks* del sistema con la ayuda de `ms2osticks()`. El valor de `CALLBACK_DELAY` es 1, el mínimo necesario para que no falle la retransmisión.

Analizando `tx()`, comienza con un cambio de estado del transceptor dejándolo en estado “*sleep*” indicado por la constante `RADIO_RST`, una demora de un milisegundo es necesaria para evitar una excepción interna en `arduino-lmic`. Luego se da la carga de datos en la variable global estructurada `LMIC`, donde en `LMIC.frame` se carga el texto a enviar mientras se actualiza el largo en `LMIC.dataLen`. Finalmente se carga la función de *callback* para cuando termine el modo TX y el transceptor se establece en modo TX con la llamada a `os_radio(RADIO_TX)`. En esencia, este es el código relevante para un *jammer* continuo. Además es necesaria una configuración extra en el archivo `project_config/lmic_project_config.h` de la librería, que involucra agregar la línea `#define DISABLE_JOIN` para evitar escribir funciones utilizadas en modo OTAA y se debe elegir la banda LoRa a utilizar, eligiendo entre incluir `#define CFG_au915 1` o `#define CFG_us915 1`.

En lo que respecta a TTNode, para facilitar la verificación de anulación de sus paquetes en las frecuencias que utiliza, se modificó la librería “TheThingsNode” descargada para Arduino IDE (referirse al Anexo B). Con los cambios aplicados se logró enviar mensajes por un canal único en modo ABP, se seleccionó el canal 8 para coincidir con el valor en el código del *jammer*.

Con el equipamiento configurado y TTNode enviando, se procedió a activar el *jammer* y observar el comportamiento del tráfico en la consola de TTN, la dinámica se muestra en la Figura 28:

	time	frequency	mod.	CR	data rate	airtime (ms)	cnt	
	▲ 19:20:00	916.8	lora	4/5	SF 7 BW 125	51.5	22	dev addr: 26 01 13 26 payload size: 19 bytes
	▲ 19:19:53	916.8	lora	4/5	SF 7 BW 125	51.5	21	dev addr: 26 01 13 26 payload size: 19 bytes
	▲ 19:19:45	916.8	lora	4/5	SF 7 BW 125	51.5	20	dev addr: 26 01 13 26 payload size: 19 bytes
JAMMING 2: ~45 segundos 5 mensajes	▲ 19:19:00	916.8	lora	4/5	SF 7 BW 125	51.5	14	dev addr: 26 01 13 26 payload size: 19 bytes
	▲ 19:18:53	916.8	lora	4/5	SF 7 BW 125	51.5	13	dev addr: 26 01 13 26 payload size: 19 bytes
	▲ 19:18:45	916.8	lora	4/5	SF 7 BW 125	51.5	12	dev addr: 26 01 13 26 payload size: 19 bytes
JAMMING 1: ~53 segundos 6 mensajes	▲ 19:18:38	916.8	lora	4/5	SF 7 BW 125	51.5	11	dev addr: 26 01 13 26 payload size: 19 bytes
	▲ 19:17:45	916.8	lora	4/5	SF 7 BW 125	51.5	4	dev addr: 26 01 13 26 payload size: 19 bytes
	▲ 19:17:38	916.8	lora	4/5	SF 7 BW 125	51.5	3	dev addr: 26 01 13 26 payload size: 19 bytes
	▲ 19:17:30	916.8	lora	4/5	SF 7 BW 125	51.5	2	dev addr: 26 01 13 26 payload size: 19 bytes
	▲ 19:17:23	916.8	lora	4/5	SF 7 BW 125	51.5	1	dev addr: 26 01 13 26 payload size: 19 bytes
	▲ 19:17:15	916.8	lora	4/5	SF 7 BW 125	51.5	0	dev addr: 26 01 13 26 payload size: 19 bytes

Figura 28: *Jamming* continuo sobre TTNode en la frecuencia 916.8 en dos intervalos.

Luego del encendido de TTNode se esperó al quinto mensaje para activar el *jammer*. El primer período fueron aproximadamente 53 segundos, observando la pérdida de 6 mensajes seguidos. Los mensajes continuaron su flujo normal y en el paquete número 15 se activó nuevamente el *jammer* por un período de aproximadamente 45 segundos, perdiéndose 5 mensajes seguidos. Finalmente, el flujo continuó en el mensaje número 21. Esta prueba demostró la viabilidad de afectar una frecuencia incluso con todos los dispositivos en distancias cortas (unos metros de distancia). Las investigaciones [27, 31] indican que un *jammer* es más efectivo si se encuentra más próximo al GW que el ED al que intenta afectar. Otro detalle interesante es que el valor de SF es 7, el mínimo en LoRa y el más dificultoso de afectar en términos de tiempo[27]. Esto es porque a mayor SF, mayor es el alcance del mensaje y la demora en transmitirse. Es decir que un SF de 7 utiliza el menor tiempo de TX posible y reduce la probabilidad de ser afectado a tiempo por *jamming*. Un SF de 12 en cambio da mayor margen a ser afectado pero la energía por símbolo de TX es necesariamente mayor y este, en el *jammer*, debe ser capaz de superarla para no ser ignorado en el destino. Sin embargo, un SF de 7 implica menor tiempo y señal para interpretar, además de menor energía utilizada, por lo que es menos robusto y más fácil de afectar.

Obviamente con un *jammer* continuo, cuanto menor sea la demora entre transmisiones menos importará el valor del SF, aunque sí importa para saber qué mensajes afectar porque el SF

permite ortogonalidad (mensajes concurrentes en un mismo canal). En [31] se cita a [56] para acotar que de todas formas el SF por sí solo no ofrece ortogonalidad, sino que depende también del valor del ancho de banda (*bandwidth*, **BW**). Se presentan las siguientes ecuaciones:

$$Symbol\ rate = \frac{BW}{2^{SF}} \quad (1)$$

$$Chirp\ rate = BW \times (Symbol\ rate) \quad (2)$$

$$\Rightarrow Chirp\ rate = \frac{BW^2}{2^{SF}} \quad (3)$$

En LoRa un *symbol* representa uno o más bits de datos, particularmente la cantidad de bits que caben en un *symbol* es el SF (en LoRa serán de 7 a 12 bits) [57], sus representaciones son líneas rectas que varían su largo y pendiente. La longitud de las rectas aumenta con su SF pues para mayor alcance se demora más en transmitir. En [56] se explica que el valor del *chirp rate* es el que determina una colisión de señales cuando el valor coincide para dos pares SF-BW. Si en la ecuación 3 se usan los valores de dos señales con los pares $\{SF = 7, BW = 125\ kHz\}$ y $\{SF = 9, BW = 250\ kHz\}$, ambos resultados son 122,0703125 y deberían colisionar. La Figura 29 muestra *symbols* no ortogonales para tres pares distintos de SF-BW y la Figura 30 muestra otros tres que sí son ortogonales (notar que se determina por la pendiente). Por lo tanto, el *jammer* continuo teóricamente podría funcionar con otros parámetros de SF y BW, pero por simplicidad y seguridad se prefirió optar por parámetros idénticos a la transmisión original²⁰. La Figura 31 muestra las combinaciones de SF y BW con su respectiva ortogonalidad.

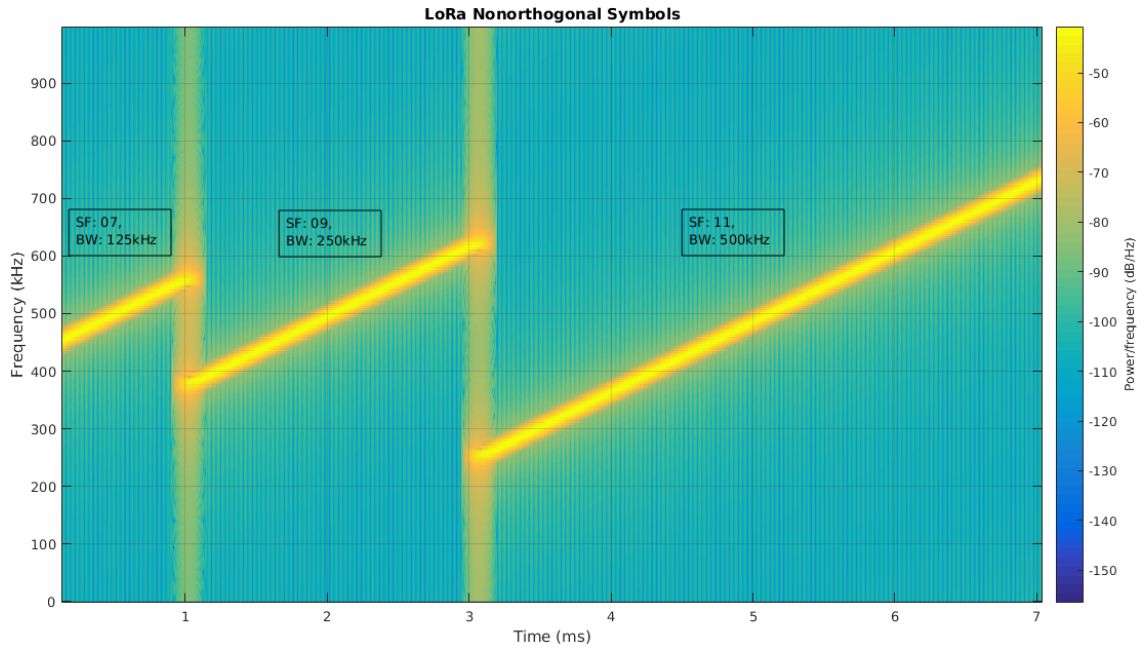


Figura 29: *Symbols* no ortogonales con diferentes SF y BW, misma pendiente [56].

²⁰Se probaron las combinaciones $\{SF = 9, BW = 250\ kHz\}$ y $\{SF = 11, BW = 500\ kHz\}$. La primera funcionó luego de algunos reposicionamientos y la segunda no logró a tener efecto en las mismas condiciones de prueba, seguramente requiera otro tipo de escenario.

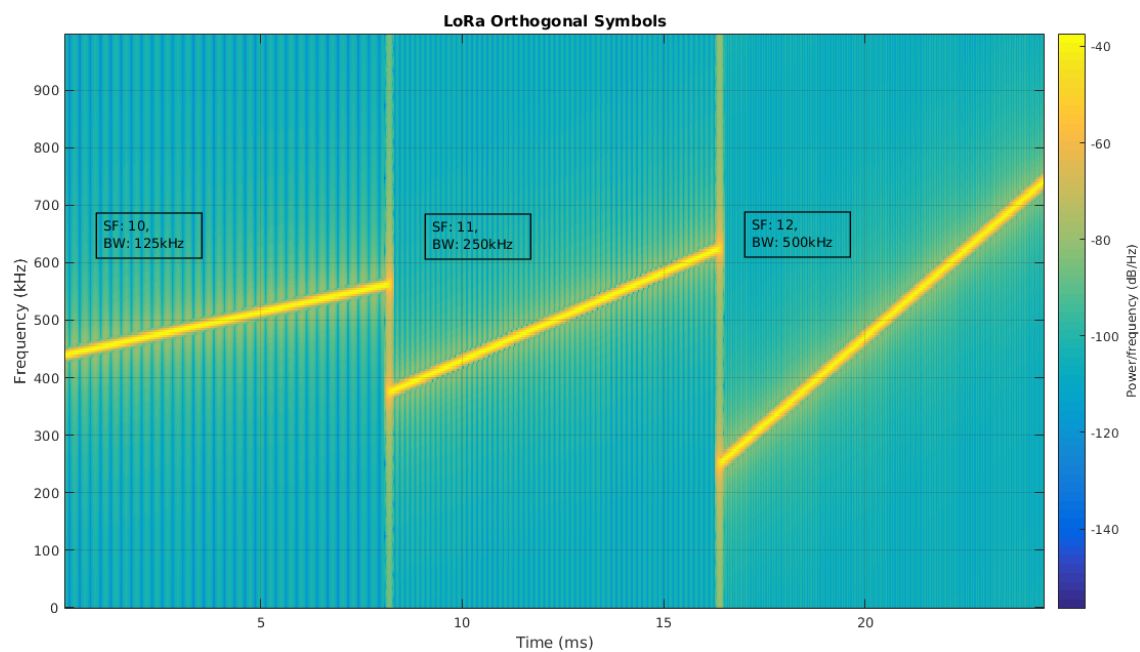


Figura 30: *Symbols* ortogonales con diferentes SF y BW, diferente pendiente [56].

SF		7	8	9	10	11	12	7	8	9	10	11	12	7	8	9	10	11	12
	BW	125	125	125	125	125	125	250	250	250	250	250	250	500	500	500	500	500	500
7	125	X								X								X	
8	125		X								X								X
9	125			X								X							
10	125				X								X						
11	125					X													
12	125						X												
7	250							X							X				
8	250								X							X			
9	250	X								X								X	
10	250		X								X								X
11	250			X								X							
12	250				X								X						
7	500													X					
8	500														X				
9	500							X								X			
10	500								X								X		
11	500	X								X								X	
12	500		X								X								X

Figura 31: Ortogonalidad de combinaciones entre SF y BW, las “X” indican no ortogonalidad.

4.4.3. *Jamming* reactivo

A diferencia del continuo, el *jamming* reactivo, disparado o *triggered* inicia cuando se detecta actividad en el canal. Es más eficiente en términos de consumo y es el paso previo a un *jammer* selectivo. La implementación generada también se basa en el ejemplo `raw.ino` y se extendió el código del *jammer* continuo para coexistir con el reactivo, seleccionando el modo con una constante de C++ antes de cargarse en una SparkX. A continuación, se explica lo agregado y las pruebas realizadas.

Las funciones iniciales `setup()` y `loop()` no cambian, exceptuando que al final de la primera en lugar de ejecutar:

```
os_setCallback(&osJob, tx_func);
```

se ejecuta:

```
os_setCallback(&osJob, first_rx_func);
```

Es decir que cambia la primera tarea a encolar. En este caso la función `first_rx_func()` se encarga únicamente de iniciar el transceptor en modo RX, para quedar a la espera de algún mensaje al cual afectar e imprimir un mensaje por serial por si se estuviera leyendo en tal interfaz:

```
1 static void first_rx_func (osjob_t* job) {
2     Serial.println("Starting first RX");
3     rx(rx_func);
4 }
```

Las siguientes funciones son las que hacen posible la escucha:

```
1 // Enable rx mode and call func when a packet is received
2 void rx(osjobcb_t func) {
3     os_radio(RADIO_RST); // Stop TX first
4     delay(DELAY_FOR_RX); // Wait a bit, without this os_radio below asserts,
5     // apparently because the state hasn't changed yet
6     LMIC.osjob.func = func;
7     LMIC.rxtime = os_getTime(); // RX _now_
8     // Enable "continuous" RX (e.g. without a timeout, still stops after receiving a packet)
9     os_radio(RADIO_RXON);
10 }
11
12 static void rx_func (osjob_t* job) {
13     // Got message, trigger jamming until JAMMING_TIME passed
14     endJammingTime = os_getTime() + ms2osticks(JAMMING_TIME);
15     os_setTimedCallback(&osJob, os_getTime() + ms2osticks(CALLBACK_DELAY), tx_func);
16     ...
```



```

17     printBytes();
18     ...
19 }

```

La primera de las funciones en ser invocada es `rx()`, análoga a `tx()` descrita en 4.4.2, con la diferencia de que comienza con una demora de `DELAY_FOR_RX` luego del *reset* del transceptor. Este valor es arbitrario, definible por código y se buscó el menor valor que permitiera hacer funcionar el modo RX, que resultó ser 37. Cuando se probó en el valor inicial 1 (y subsiguientes) el transceptor permanecía en un estado inconsistente sin efectuar RX ni TX. Otra diferencia es la asignación del campo `LMIC.rxtime` con el tiempo actual del sistema para iniciar RX inmediatamente y que la radio se pone en modo RX con `os_radio(RADIO_RXON)`. Cuando un mensaje es recibido, `arduino-lmic` invoca a la función asignada en `LMIC.osjob.func`, en este caso a `rx_func()`. La lógica se da en dos líneas (14 y 15), siendo la primera un cálculo de tiempo hacia el futuro (tiempo actual sumado a `JAMMING_TIME`) que determina el fin del *jamming* y la segunda un encolamiento de la función `tx_func()` para iniciar la transmisión. Además se incluye una invocación a `printBytes()` que muestra en formato hexadecimal el mensaje recibido, el *payload* físico descrito en 2.3.3.4. Se comenta sobre esto más adelante.

Respecto a la transmisión, la función `tx()` permanece incambiada. Sin embargo `tx_func()` tiene una modificación para el caso de *jammer* reactivo:

```

1  static void tx_func (osjob_t* job) {
2      tx("Jamming", txdone_func);
3      ...
4      // Reschedule job every CALLBACK_DELAY until reaching end jamming time
5      if (os_getTime() > endJammingTime) {
6          ...
7          rx(rx_func);
8      } else {
9          os_setTimedCallback(job, os_getTime() + ms2osticks(CALLBACK_DELAY), tx_func);
10     }
11     ...
12 }

```

En *jamming* continuo la única acción luego de llamar a `tx()` es encolar la siguiente transmisión con `os_setTimedCallback()` como en la línea 9. En este caso primero se debe controlar si se cumplió el tiempo total de *jamming* consultando el tiempo actual y comparándolo con la variable global `endJammingTime` asignada en `rx_func()`. Si se cumple, se vuelve al modo RX y el ciclo se repite, pero si no se cumple se vuelve a transmitir.

En el mismo escenario del *jamming* continuo se encendió el TTNode para ver los mensajes iniciales. En la interfaz serial del SparkX se vio el siguiente texto impreso:

```
402613012600070002527EC7FB096909D34EA9
```

Luego el *jammer* se puso en modo TX, pero el mensaje no fue afectado. Se cambiaron las distancias y posiciones de los dispositivos, pero nada cambió. Analizando el primer mensaje

recibido, se comparó con el formato de la especificación de LoRaWAN, desglosándolo hasta identificar las partes de largo fijo:

```
Physical: Preamble|PHYHDR|PHYHDR_CRC|PHYPayload|CRC
PHYPayload: MHDR(1)|MACPayload(1..M)|MIC(4)
MHDR: MType(3 bits)|RFU(3 bits)|Major(2 bits)
MACPayload: FHDR|FPort|FRMPayload
FHDR: DevAddr(4)|FCtrl(1)|FCnt(2)|FOpts(0..15)
```

Considerando que sería un PHYPayload se analizó el comienzo, el primer byte, correspondiente al cabezal MAC o MHDR. El primer byte recibido es 0x40, siendo en binario 01000000. Separando los bits como se indica más arriba deberían ser MType = 010, RFU = 000 (*Reserved for Future Usage*) y Major = 00. Según [21] estos valores corresponden a un MType de “Unconfirmed Data Up” (un *uplink* de datos común, sin confirmación requerida) y un Major asociado a LoRaWAN R1 (el único valor provisto por la especificación), por lo que coincide perfectamente con un MHDR. Aún así se observó el inicio de lo que sería el MACPayload que comienza con 4 bytes fijos de DevAddr. Los siguientes 4 bytes recibidos son 0x26130126 y deberían coincidir con el DevAddr de la Figura 28, es decir 26011326. A primera vista no lo hacen, pero separando de a byte, es claro que los bytes se ordenaron en formato *little endian*, quedando en reversa. Así se confirmó que efectivamente se recibe un PHYPayload congruente con las observaciones de la consola de TTN.

En intentos posteriores por reducir el tiempo de procesamiento de RX en `arduino-lmic` se trató de reducir en la librería el tamaño máximo de mensajes con la esperanza de que se truncara la recepción y se retornaran los datos recibidos hasta ese instante, pero no hubo éxito pues el comportamiento interno es no activar la interrupción de fin de RX si no se toma el mensaje completo. Este es el mayor problema del *jammer* reactivo porque idealmente se decide activar apenas es detectado el preámbulo del mensaje, cualquier demora extra puede invalidar completamente el *jamming*. Aquí la demora es total porque no se logró terminar la recepción sin leer todo el mensaje. A pesar de que no se logró el objetivo a corta distancia, se dio un siguiente paso en la experimentación para llevarlo a distancias mayores con el fin de observar si alejar el ED se transforma en ventaja de tiempo para el *jammer* (suponiendo que la distancia sumaba alguna demora que alcanzara para afectar sólo el final de la señal legítima).

Con esto en mente se dejó el GW en una posición fija y se alejaron el ED y el *jammer* a distancias que aproximadamente intentaron mantener línea recta con los menores obstáculos posibles. El TTNNode es el que más alejado se ubicó del GW y se verificó en cada avance de distancia, en tiempo real, que los mensajes siguieran siendo recibidos por el GW para evitar falsos positivos de *jamming*. Cada 100 metros avanzados se procedía a encender el *jammer* y controlar si se perdían mensajes (la posición del *jammer* varió constantemente en el experimento, posicionado en diferentes puntos entre TTNNode y GW). A los 300 metros aproximadamente se detectó una pérdida de 4 paquetes seguidos con el *jammer* a aproximadamente 100 metros del TTNNode, sin embargo, cuando se intentó reproducir (apagando y encendiendo nuevamente a los 30 segundos) la pérdida no volvió a ocurrir. Esto podría deberse a un falso positivo relacionado con factores ambientales (presencia de niebla y circulación de peatones y vehículos en la trayectoria de la señal) o quizás

se dio una condición específica en la que el *jamming* tuvo efecto. Lamentablemente no pudo ser validado con mayor nivel de certeza. El experimento finalizó aproximadamente a los 520 metros, siendo que no hubo resultados comprobables y el alcance del TTNode se tornó desfavorable (se perdían mensajes). Se concluyó que para ver resultados favorables con la implementación planteada de *jammer* reactivo sería necesario mejor equipamiento para incrementar las distancias y energía de las señales, siendo alternativas utilizar otra implementación o investigar en profundidad la manipulación del *hardware* y la librería `arduino-lmic`.

Sin éxito en el experimento, se decidió no avanzar en mejorar el código para obtener un *jammer* selectivo y menos aun sumándole un *wormhole*. Sin embargo, se comprobó que leyendo los bytes recibidos es posible distinguir el tipo de mensaje y el DevAddr, valores para potenciales criterios de selección para el *jamming* selectivo. Claramente, cuanto más profundo se lea en los bytes (si se hubiera logrado detener la recepción antes) menos margen de tiempo tiene el *jammer* para activarse y afectar el mensaje detectado.

4.5. Modificación de mensajes

Con un *jammer* continuo funcional el siguiente paso para cubrir más de los árboles de ataque es lograr recibir mensajes, modificarlos y reenviarlos. Para evitar que lleguen al NS mensajes con idéntico FCnt y se convierta en una condición de carrera (uno será descartado), es que entra en juego el *jammer*. El objetivo final es construir una prueba de concepto que conste de dos ED maliciosos: un *jammer* y una suerte de implementación de MitM (*Man-in-the-Middle*), se le asignan los nombres M1 y M2 respectivamente para la siguiente explicación. El flujo se da de la siguiente manera:

1. El ED envía su mensaje.
2. M1 se encuentra activo en el mismo canal (más específicamente, mismo *chirp rate*). Por lo que el mensaje será afectado y el NS no podrá validar su integridad correctamente, descartándolo.
3. A la vez, M2 se encuentra lo suficientemente cerca del ED pero lo suficientemente lejos (o estratégicamente posicionado aunque esté cerca) de M1, para que pueda recibir el mensaje original completo.
4. Con las claves de sesión LoRaWAN, M2 descifra el mensaje original, lo modifica a criterio y con las claves nuevamente lo cifra conservando el valor de FCnt.
5. M2 envía el mensaje modificado por un canal distinto al original, no afectado por el *jamming*.
6. El NS recibe el mensaje modificado, lo valida y lo envía al AS para ser procesado.

4.5.1. Construcción

En primer lugar, debía construirse una prueba de concepto que lograra el paso 4 del flujo mencionado. Se buscaron implementaciones en C++ que pudieran integrarse a la solución actual

con `arduino-lmic` y la seleccionada fue `util-lorawan-packets` [58]. A pesar de no tener documentación sobre su uso ni ejemplos, es una librería relativamente fácil de leer e interpretar, con una API descriptiva en los nombres de sus funciones. A prueba y error en pocos intentos se obtuvo una prueba de concepto funcional, aunque fue necesario editar la librería, ediciones documentadas en el Anexo C. Se comentan las partes relevantes del código que se le suman o modifican al código del *jammer*, comenzando por la inicialización:

```

1  uint8_t nwksKey[16] = { ... };
2  uint8_t appSKey[16] = { ... };
3  uint8_t devEUI[8] = { ... };
4  lwPackets_api_t api;
5  lwPackets_state_t state;
6
7  void setupLw() {
8      api.LogError = logLw;
9      api.LogInfo = logLw;
10     // Lib uses default implementations when pointers are NULL
11     api.free = nullptr;
12     api.malloc = nullptr;
13     state.pDevCfg = new Lorawan_devCfg_t();
14     state.pFCntCtrl = new Lorawan_fcnt_t();
15     state.pFCntCtrl->FCntUp = 0;
16     // Set all types of NwkKeys just in case
17     loadNwkSKey(state.pDevCfg->NwkSEncKey);
18     loadNwkSKey(state.pDevCfg->FNwkSIntKey);
19     loadNwkSKey(state.pDevCfg->SNwkSIntKey);
20     loadAppSKey(state.pDevCfg->AppSKey);
21     loadDevEUI(state.pDevCfg->DevEUI);
22     state.pDevCfg->LorawanVersion = LORAWAN_VERSION_1_0;
23     LoRaWAN_PacketsUtil_Init(api, state);
24 }
```

Antes de comenzar a utilizar la librería, `LoRaWAN_PacketsUtil_Init()` debe ser llamada para inicializarla, pasándole por parámetro una estructura de tipo `lwPackets_api_t` que contiene punteros a funciones a utilizar internamente y una estructura de tipo `lwPackets_state_t` con información vital para el objetivo consumida internamente por las funciones que se comentarán al describir la recepción. Se creó la función `setupLw()` que encapsula la inicialización. Se comienza asignando a `api` las implementaciones para registro de mensajes (la definición de `logLw()` no es relevante, un ejemplo se encuentra comentado en el código de la librería) y para liberación/reserva de memoria. Cuando el valor es un puntero nulo, la librería usa implementaciones por defecto. Luego los datos básicos en `state` son un FCnt inicial (0) que se utiliza internamente, las claves de sesión²¹ y DevEUI (las funciones `loadNwkSKey()`, `loadAppSKey()` y `loadDevEUI()` cargan

²¹La `NwkSKey` se asigna a tres claves de la librería que según sus nombres deberían corresponderse a claves en LoRaWAN 1.1. Sin embargo, necesitan tener un valor válido para armar/desarmar mensajes. En particular y según el código, `NwkSEncKey` debería servir solo para casos de comandos MAC, sin *payload* en el mensaje.

los arreglos correspondientes de las líneas 1-3) y por último la versión de LoRaWAN que en este caso es `LORAWAN_VERSION_1_0`. Debería soportar también 1.1 pero no fue probado. El contenido de `setup()` es idéntico al *jammer* reactivo con la salvedad de que incluye `setupLw()` al inicio.

Como comienza de la misma forma que el *jammer* reactivo, se inicia en RX por el canal 8 hasta recibir un mensaje (las funciones `rx()` y `tx()` no cambian), momento en el que se invoca a `rx_func()`. El siguiente código muestra el resto de las funciones relevantes:

```
1  static void txdone_func (osjob_t* job) {
2      ...
3      // RX when TX is done
4      uint8_t uplinkChannel = 8;
5      setupLMICChannel(uplinkChannel);
6      rx(rx_func);
7  }
8
9  static void tx_func (osjob_t* job) {
10     // TX in different channel
11     uint8_t uplinkChannel = 9;
12     setupLMICChannel(uplinkChannel);
13     tx(txdone_func);
14 }
15
16 static void rx_func (osjob_t* job) {
17     ...
18     lorawan_packet_t* decodedPacket = LoRaWAN_UnmarshalPacket(LMIC.frame, LMIC.dataLen);
19
20     // Assuming this is the decrypted payload data, substitute content
21     ...
22     decodedPacket->pPayload[0] = 0x31;
23     ...
24     // Update library's internal FCntUp with the one in the packet
25     state.pFCntCtrl->FCntUp = decodedPacket->BODY.MACPayload.FHDR.FCnt16;
26
27     // Now make the packet again
28     LoRaWAN_MarshalPacket(decodedPacket, (uint8_t*)&LMIC.frame, LMIC.dataLen);
29     ...
30     // Delete temporal decoded packet
31     LoRaWAN_DeletePacket(decodedPacket);
32     // Queue TX
33     os_setTimedCallback(job, os_getTime() + ms2osticks(CALLBACK_DELAY), tx_func);
34 }
```

En `rx_func()` entra en acción la nueva librería. El primer paso es descifrar el mensaje contenido en `LMIC.frame`, así que en la línea 18 se invoca a `LoRaWAN_UnmarshalPacket()` indicando

además el largo del mensaje. Internamente la librería intentará desarmar (*marshal* hace referencia a armado, alineamiento o reunión) el mensaje para detectar sus partes, validar la integridad con el MIC y la clave NwkSKey, descifrar el *payload* con la clave AppSKey y devolver el resultado estructurado en un puntero de tipo `lorawan_packet_t` definido en la librería. Si nada falla, en el campo `pPayload` se encuentra el *payload* descifrado y como se ve en la línea 22, es accesible como arreglo para modificar uno de sus bytes.

Luego es muy importante actualizar el valor de `FCntUp` en la estructura `state` de la librería (línea 25) porque internamente se utiliza para validar el MIC en mensajes posteriores (tanto en *unmarshal* como en *marshal*) y mantiene el contador como si fuera un NS. Teniendo el `FCntUp` actualizado y el mensaje modificado se procede al armado de un nuevo mensaje con `LoRaWAN_MarshalPacket()` pasándole el nuevo mensaje, el *buffer* donde se almacenarán los bytes del mensaje en crudo (`LIMC.frame`) y la longitud resultante (`LIMC.dataLen`). Finalmente, se debe destruir la estructura del mensaje para evitar problemas de memoria con `LoRaWAN_DeletePacket()` y se procede a activar la transmisión con `os_setTimedCallback()` como en los ejemplos de código anteriores.

Antes que la transmisión comience, en `tx_func()` se configura el canal fijo 9 para enviar asegurando que es distinto al que se utilizó para recibir. Cuando la transmisión finaliza, como se ve en `txdone_func()`, el canal vuelve a ser el 8 y se activa nuevamente el modo RX, iniciando el ciclo ya descrito.

4.5.2. Ejecución

Con el código completo se procedió a una prueba integrando todos los componentes: *jammer* SparkX, TTNode, TTGW y SparkX con el nuevo código. Los dispositivos siguieron más de una distribución mientras se probaban distancias que no afectarían negativamente a algún componente, sin embargo, el escenario final fue el de la Figura 32. Los pasos seguidos en la prueba fueron los siguientes:

1. Encender TTNode.
2. Luego del mensaje 2 encender el SparkX que implementa MitM. Desde este punto se observan mensajes con `FCnt` idéntico, correspondientes al mensaje original de TTNode en la frecuencia 916.8 y al mensaje malicioso del SparkX en la frecuencia 917 (ver Figura 33).
3. Desde el mensaje con `FCnt` 6, el *jammer* es activado en la frecuencia 916.8, corrompiendo los mensajes de TTNode. Aunque las distancias son tales que permiten al segundo SparkX recibir sin interferencias los mensajes originales.

El ataque se considera efectivo en el momento que solo los mensajes del dispositivo impostor son recibidos en TTGW. Antes de este punto sería sospechoso en caso de monitorear el tráfico y encontrar los mensajes con `FCnt` duplicado. Las sospechas pueden ser reales, aunque en implementaciones inconsistentes podrían enviarse mensajes múltiples con un mismo `FCnt`, pero distintas frecuencias.

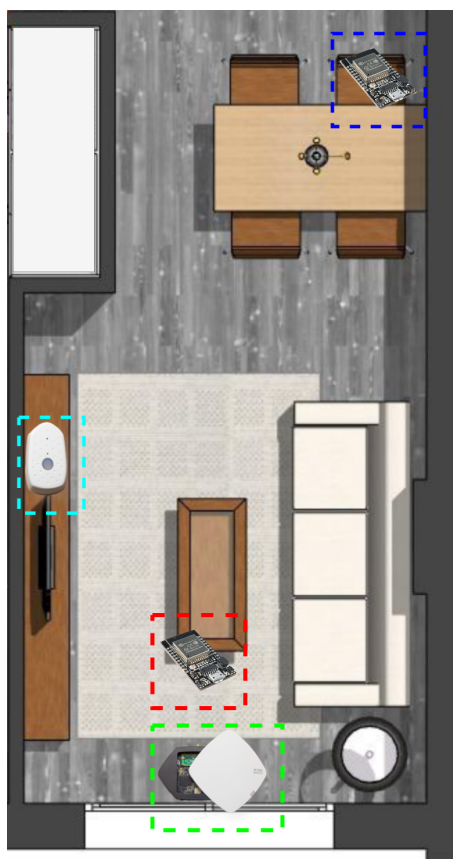


Figura 32: Escenario para prueba completa de modificación de mensajes. Azul: SparkX implementando recepción, modificación y envío de mensajes. Celeste: TTNode enviando mensajes periódicos. Rojo: SparkX oficiando de *jammer*. Verde: TTGW.

uplink		downlink	join	0 bytes		×	▶ resume		🗑 clear
time	frequency	mod.	CR	data rate	airtime (ms)	cnt			
▲ 00:13:55	917	lor	4/5	SF 7 BW 125	51.5	9	dev addr: 26 01 1B E1	payload size: 19 bytes	
▲ 00:13:48	917	lor	4/5	SF 7 BW 125	51.5	8	dev addr: 26 01 1B E1	payload size: 19 bytes	
▲ 00:13:41	917	lor	4/5	SF 7 BW 125	51.5	7	dev addr: 26 01 1B E1	payload size: 19 bytes	
▲ 00:13:33	917	lor	4/5	SF 7 BW 125	51.5	6	dev addr: 26 01 1B E1	payload size: 19 bytes	
Jammer activo									
▲ 00:13:26	917	lor	4/5	SF 7 BW 125	51.5	5	dev addr: 26 01 1B E1	payload size: 19 bytes	
▲ 00:13:25	916.8	lor	4/5	SF 7 BW 125	51.5	5	dev addr: 26 01 1B E1	payload size: 19 bytes	
▲ 00:13:18	917	lor	4/5	SF 7 BW 125	51.5	4	dev addr: 26 01 1B E1	payload size: 19 bytes	
▲ 00:13:18	916.8	lor	4/5	SF 7 BW 125	51.5	4	dev addr: 26 01 1B E1	payload size: 19 bytes	
▲ 00:13:11	917	lor	4/5	SF 7 BW 125	51.5	3	dev addr: 26 01 1B E1	payload size: 19 bytes	
MitM activo									
▲ 00:13:10	916.8	lor	4/5	SF 7 BW 125	51.5	3	dev addr: 26 01 1B E1	payload size: 19 bytes	
▲ 00:13:03	916.8	lor	4/5	SF 7 BW 125	51.5	2	dev addr: 26 01 1B E1	payload size: 19 bytes	
▲ 00:12:55	916.8	lor	4/5	SF 7 BW 125	51.5	1	dev addr: 26 01 1B E1	payload size: 19 bytes	
▲ 00:12:48	916.8	lor	4/5	SF 7 BW 125	51.5	0	dev addr: 26 01 1B E1	payload size: 19 bytes	

Figura 33: Flujo capturado de mensajes en TTGW con el escenario planteado.

5. Contramedidas

Con una amplia superficie cubierta en términos de potenciales amenazas y ataques en la familia de versiones LoRaWAN 1.0.*, se concluye el trabajo proporcionando posibles soluciones a estos problemas. La mayoría de estas son teóricas y algunas se basan en la experiencia resultante de la experimentación. A continuación, se presentan dos subsecciones. La primera refiere a la importancia de la auditoría en LoRaWAN, mencionando la herramienta para esta tarea que ofrece el *framework* LAF [52] ya mencionado en este informe. La segunda busca listar los diferentes ataques identificados junto con las recomendaciones pertinentes para mitigarlos o prevenirlos.

5.1. Auditoría

En el *whitepaper* del *framework* LAF [22] se proponen aplicar controles que tienen como objetivo agregar una capa de seguridad pasiva en una red LoRaWAN. Por ejemplo, si bien no es posible detectar el momento exacto en el que la AppKey de un dispositivo es comprometida, se propone aplicar controles que permitan inferir si una red LoRaWAN está siendo atacada por un hacker malicioso, o detectar si la red ya se encuentra comprometida. Asimismo, otros controles que se proponen en [22] tienen como objetivo detectar problemas de mal funcionamiento o errores de configuración. En el resto de esta sección se presentan y discuten estos controles.

El *framework* LAF y sus herramientas están implementados en una serie de *scripts* que serán referenciados cuando se expliquen, mencionando su ubicación en el repositorio de GitHub [52]. En las siguientes secciones se omiten los detalles completos de invocación de cada herramienta/*script*, puesto que esta información ya se encuentra en la misma documentación del repositorio en el archivo README.md.

Para poder analizar es necesario almacenar los mensajes recibidos, y LAF provee para esto una serie de *scripts* que reciben paquetes LoRaWAN de diferentes fuentes y los guardan en archivos, con un formato estándar. Estos archivos luego pueden ser procesados por el *script* en `/auditing/analyzers/LafProcessData.py` para ejecutar diferentes herramientas de análisis. Estos *scripts* están bajo `/auditing/datacollectors` y son:

- `GenericMqttCollector.py` - Se conecta a un *broker* MQTT.
- `LoraServerIOCollector.py` - Se conecta a un LoRaServer.
- `PacketForwarderCollector.py` - Recibe paquetes desde un *forwarder* y los persiste.
- `TTNCollector.py` - Se conecta a The Things Network.

Entonces luego el *script* `analyzers/LafProcessData.py` se puede ejecutar sobre los paquetes recibidos que dependiendo de las opciones indicadas, analizará el tráfico y/o intentará hacer fuerza bruta de la AppKey. Las diferentes alertas que LAF devuelve del análisis se explican en las siguientes secciones.

5.1.1. Análisis realizados por LAF

5.1.1.1 *Replay* de mensajes

Mediante el uso de una base de datos que almacene los mensajes recibidos, es posible controlar si un mensaje ya fue recibido. La mejor manera de hacer esto es controlando el MIC del mensaje. En caso de que dos mensajes recibidos tengan un MIC idéntico, el resto de los campos como el tipo de mensaje (MType), DevAddr y DevEUI pueden revisarse. Este enfoque permite detectar rápidamente mensajes duplicados, evitando tener que controlar campo por campo, lo que no es razonable para un gran volumen de mensajes.

Un método alternativo para el caso particular de detectar mensajes de JR repetidos, consiste en buscar DevNonces duplicados para el mismo dispositivo, este campo debería de ser suficientemente aleatorio para no colisionar con frecuencia. En LAF este control se identifica como **LAF-001**.

5.1.1.2 Mensajes falsos y denegación de servicio (sesiones simultáneas)

En el caso de OTAA, una vez que un atacante obtiene la AppKey, puede generar claves de sesión e inyectar información falsa en el NS si se cumple la siguiente condición: el campo FCnt del mensaje tiene que ser mayor que el valor FCnt del último mensaje recibido por el NS. Entonces si el atacante logra enviar un mensaje con un valor mayor de FCnt y si el dispositivo legítimo continúa enviando mensajes (funcionando de manera normal), el NS empezará a descartar mensajes válidos, debido a que van a tener un FCnt menor al último recibido. Debido a esto, cuando el NS recibe un mensaje con un valor de FCnt menor al esperado, es posible inferir que una sesión paralela fue establecida para ese dispositivo y alguien está realizando un ataque de denegación de servicio o enviando mensajes falsos.

Para esta sección, el único control que sí está implementado es **LAF-007** que detecta si se recibió un FCnt menor al esperado (y distinto de 0, para descartar reinicios). Si bien en la documentación de LAF se identifican otros controles asociados identificados **LAF-003** (*replay* de JR), **LAF-004** (mensajes duplicados de *uplink*) y **LAF-005** (mensajes duplicados de *downlink*), aún no han sido implementados en el *framework*.

5.1.1.3 Dispositivos ABP

Según [22], se menciona que implementar dispositivos usando ABP no es una buena práctica de seguridad y está desaconsejado, porque que las claves de sesión son las mismas durante todo el ciclo de vida del dispositivo. Se considera que este enfoque puede ser un poco extremo ya que, si bien las justificaciones que lo respaldan son razonables, aún se puede utilizar ABP si se conocen y toman medidas para mitigar los riesgos asociados.

Como los desarrolladores de LAF aconsejan no usar ABP, proveen un control para identificar dispositivos que lo hagan (con el fin de luego desactivarlos o cambiarlos a OTAA). Se referencia con **LAF-006** y se analiza si un contador se reinicia (FCnt pasa a valer 0), se mantiene el

DevAddr y no se detecta ningún *Join Procedure* previo. Se infiere entonces que el dispositivo es ABP.

5.1.1.4 Claves conocidas o no suficientemente aleatorias

Es un problema tener claves que sean fáciles de adivinar o se puedan conocer (por defecto, impresas en el dispositivo, etc.). Para identificar la presencia de claves inseguras, se puede usar varios métodos en conjunto con las herramientas de LAF.

Los controles asociados se identifican con **LAF-008** cuando se logra generar el MIC de un JR usando una AppKey conocida y **LAF-009** cuando se adivina la AppKey usando un *string* conocido o no aleatorio. También se pueden hacer pruebas manuales de *cracking* con un paquete capturado mediante el *script* en LAF para obtener las claves de OTAA, que se encuentra en `lorawan/BruteForcer.py`. Recibe como parámetro de entrada un mensaje JA o JR que debe estar codificado en base 64 y se trata de obtener su AppKey correspondiente usando una serie de claves provistas y opcionalmente otras generadas en el momento a partir de las primeras.

5.1.1.5 Otros controles

Adicionalmente LAF implementa otros controles como **LAF-002**, que no es una amenaza de seguridad externa, pero sí del despliegue, el cual identifica que existen dos dispositivos con la misma DevAddr (pero distinto DevEUI). El problema que genera esto es que no es posible identificar en el NS desde qué dispositivo provino el mensaje. Para este caso se recomienda que, si el dispositivo fue activado por OTAA, entonces se revise la lógica usada para asignar DevAddrs y asegurarse que no se asigne el mismo DevAddr a dos dispositivos. Para el caso de ABP, controlar que el DevAddr configurado en el *firmware* del dispositivo sea único en toda la red LoRaWAN.

Por último, está el control **LAF-010** orientado a la seguridad de los GW, que informa cuando el GW cambia su ubicación, útil para detectar casos donde fue robado, movido o se está haciendo un ataque de suplantación de identidad.

5.1.2. Prueba de uso de LAF

Se realizó una prueba del *framework*, mediante la generación de diferentes paquetes LoRaWAN con el propósito de verificar el correcto funcionamiento de los controles que implementa.

Se recuerda que los controles LAF-003 a LAF-005 no están aún implementados en LAF (y por lo tanto no es posible probarlos). Por otro lado, debido a limitaciones del GW utilizado para las pruebas, no fue posible verificar los controles asociados a dispositivos usando OTAA, puesto que el GW no capturaba los mensajes de JR, estos controles son LAF-001, LAF-002, LAF-008 y LAF-009. Esto redujo a que los dos controles que fueran posibles de probar fueran LAF-006 y LAF-007 que se explican a continuación.

Como se mencionó anteriormente, el *script* para analizar los paquetes de LAF se encuentra en `/auditing/analyzers/LafProcessData.py` y se puede invocar con la *flag* `-h` para obtener una ayuda de su uso y las opciones que soporta. Entre ellas se puede indicar las pruebas a aplicar, con

`-a` se analiza el tráfico y con `-b` se realizan pruebas de fuerza bruta donde además con `-k` se puede indicar un archivo de claves a probar. También permite acotar los mensajes a analizar filtrando por su ID (`--from-id` para el ID inicial y `--to-id` el ID final), entre otras opciones. Para probar su uso se invocó usando la *flag* `-a` para solo analizar el tráfico, sin las pruebas de fuerza bruta.

Para el caso de de LAF-006 la prueba consistió en simplemente activar un ED, aguardar que se envíen algunos mensajes y luego apagarlo y volverlo a prender. En este caso LAF analizó estos paquetes e identificó que el último contador recibido era 80 y pasó a valor 0 en un momento dado, sin tener mensajes de JR/JA de por medio, dando fuertes indicios de que el ED asociado usa ABP (Figura 34).

```
DEBUG:root:Using packet: 672
DEBUG:root:LAF-006-DevAddr 26011062 counter was reset. Previous counter was 80 and received 0. This device may be ABP or device is not rejoining a counter overflow. Previous packet 152, current packet 672. Data collector ttn-gw-iot-2 (ID 1).
DEBUG:root:Using packet: 673
```

Figura 34: Alerta LAF-006 - Detección de un dispositivo ABP.

En el caso del control LAF-007, se configuró un ED enviando mensajes periódicamente y además se utilizó una de las placas SparkX con una versión levemente modificada del *jammer* reactivo (ver sección 4.4.3) que envía un mismo mensaje predefinido, usando las mismas claves que el ED y aumentando en uno el contador de este en cada envío de mensaje. Se inició con un contador menor al último enviado por el ED y como se puede ver en la Figura 35, esto generó que se detecte la alerta que indica un posible compromiso de las claves del dispositivo.

```
DEBUG:root:Using packet: 795
DEBUG:root:Using packet: 796
DEBUG:root:LAF-007-Received smaller counter for DevAddr 26011062. Previous counter was 2000 and current 51. Previous packet 795, current packet 796. Data collector ttn-gw-iot-2 (ID 1).
DEBUG:root:Using packet: 797
DEBUG:root:Using packet: 798
DEBUG:root:Using packet: 799
DEBUG:root:LAF-007-Received smaller counter for DevAddr 26011062. Previous counter was 2002 and current 52. Previous packet 798, current packet 799. Data collector ttn-gw-iot-2 (ID 1).
DEBUG:root:Using packet: 800
DEBUG:root:No more packets to process. Sleeping a while
```

Figura 35: Alerta LAF-007 - Contador recibido menor que el esperado.

Se puede ver que hay dos alertas donde en ambas se indica que se recibió un paquete con un contador menor al último recibido (y diferente de 0).

Una última aclaración importante para el uso de la herramienta es que la salida de su ejecución devuelve un mensaje por cada paquete procesado, como se puede ver en las figuras anteriores, aún si el paquete no es relevante a ninguna de las pruebas LAF-XXX. Esto hace que, para un gran volumen de paquetes a procesar, sea crítico considerar de antemano como se realizará el procesamiento de esta salida. Una sugerencia puede ser hacer uso de las *flags* que permite filtrar por el ID del paquete, limitándose a cierto rango conocido de paquetes que interesen procesar. Otra posibilidad es utilizar un comando como `grep` para filtrar la salida, y solo mostrar los paquetes que contengan los caracteres “LAF-”, devolviendo así únicamente las alertas que

corresponden a algunas de las pruebas que implementa el *framework*.

5.1.3. Conclusiones sobre LAF

Se concluye que el beneficio de hacer uso del *framework* LAF, será mayor si los dispositivos utilizados son activados con OTAA, puesto que la mayoría de los controles de seguridad implementados están orientados a este tipo de dispositivos.

De todas formas, se aclara que si bien actualmente LAF es la única herramienta encontrada que realiza controles de seguridad en LoRaWAN, considerando su estado de desarrollo está aún en fases muy iniciales (los mismos creadores indican que es una versión alfa) y no existe un desarrollo continuado de esta por el momento (revisando las estadísticas en GitHub se puede ver que fue publicada en Agosto de 2019, por el momento solo tiene 46 *commits* y desde enero de 2020 no ha tenido nuevos cambios). Se debe esperar a nuevas actualizaciones y que se encuentre en un estado más maduro de desarrollo para su uso. Por eso se considera que se debe tomar este *framework* como una referencia o punto de partida, no como una solución completa.

En lo que respecta a el *stack* tecnológico usado por LAF, se considera que tiene ciertas oportunidades de mejora. En particular respecto al almacenamiento, sería deseable hacer uso de una base de datos para persistir los mensajes que sea más adecuada para este tipo de información²², algunas de las opciones pueden ser Elasticsearch [59] o InfluxDB [60]. Para lograr esto se podrían utilizar los *scripts* que LAF ya provee para capturar y enviar los mensajes a la base de datos que se desee. Además, se sugiere la centralización de esta información para tener una mejor visión de seguridad con un *software* de gestión y análisis de logs como Graylog [61].

Además de las mejoras en el desarrollo de LAF, también sería esperable que a corto o mediano plazo surgieran productos de *software* similares, considerando que su presentación a la comunidad de LoRaWAN tuvo ciertas críticas [62], pero también fue recibido como una oportunidad de mejora para la seguridad de LoRaWAN [63].

5.2. Recomendaciones

La discriminación de los ataques en esta sub-sección no es hoja por hoja, sino que se recorren los árboles de 3.5.2 buscando los nodos que engloben efectivamente los distintos requerimientos de ataques para la explicación. Se señala que es de importancia que la organización y el administrador de la red LoRaWAN **evalúen** las recomendaciones que aquí se plantean (y sin limitarse solo a ellas, consultando la literatura referenciada) contrastando con el plan o despliegue existente y actuar en consecuencia.

5.2.1. *Bit-flipping* en enlace ED-GW

De por sí el *bit-flipping* con conocimiento y control de qué bit se cambia sin posesión de NwkSKey es muy poco probable. El factor bajo dominio del administrador de la red es que el MIC de los mensajes se controle, que es lo habitual en las implementaciones populares. Cubriendo este punto

²²Esto es para mejorar la fiabilidad de almacenamiento y gestión de los registros (comparando una base de datos con un simple archivo de texto), además de lo relacionado al acceso a la información.

el ataque ya no se puede lograr, se necesita ir un paso más allá y convertirlo en la suplantación de ED.

5.2.2. *Jamming*

Por diseño, el *jamming* en LoRa es inevitable. No se encontró literatura que indique cómo detectar que el canal está siendo ocupado para LoRa específicamente. Sin embargo, existen artículos en Internet [64] que explican a alto nivel cómo construir un detector de *jamming* genérico y sugiere equipamiento y *software*. Sí se encuentran investigaciones que mencionan que *jamming*s como el continuo son fáciles de detectar, pero sin más detalles. Por lo tanto, las siguientes recomendaciones [27] son mitigaciones basadas en la influencia que puede tener el *jamming* en el tráfico, es decir, que se deducen del comportamiento visible en el flujo de mensajes.

- **Crear redes LoRa densas con regiones de cobertura superpuestas:** Distribuir GWs distantes entre sí de forma que cubran zonas y que a su vez los ED puedan ser escuchados por más de un GW, son factores vitales para combatir el *jamming*. Para que un mensaje sea descartado, el *jammer* precisa que el RSSI de su señal sea mayor a la del ED y para esto en general es necesaria la proximidad con el GW. De haber más de un GW, el atacante debe geolocalizar a los GWs para posicionarse estratégicamente o elevar su costo en equipamiento y mejorar cobertura (con un *jammer* más potente o múltiples dispositivos).
- **Maximizar el uso de *channel hopping*:** Cambiar de canales para cada mensaje reduce las colisiones. Además, a mayor cantidad de canales distintos, más complejo debe ser el *jammer* ya que debe escuchar en todos si quiere anular completamente a un ED. Esto aumenta el presupuesto de un *hardware* LoRa de bajo costo a un dispositivo más costoso como lo es un GW multicanal. Cabe destacar que es deseable que el cambio de canales sea **aleatorio**, dado que si un patrón está presente es posible construir un ataque basado predicción [65] y ahorrar en costos para detección multicanal.
- **Moverse a un SF mayor:** En [27] se muestra empíricamente que con un SF de 12 es más difícil para el *jammer* alcanzar el RSSI necesario para crear un diferencial de dB suficiente respecto a los mensajes del ED. Por supuesto un mayor SF implica más tiempo de envío y consecuentemente más margen de reacción para el *jammer*, aunque este debe lograr alcanzar la energía necesaria en ese intervalo (reduciendo por ejemplo su distancia al GW). Cabe destacar que en las regiones donde existe *duty cycle*, aumentar el SF reduce los límites permitidos para mensajes legítimos, que a la fecha no es el caso en Uruguay.
- **Vencer el tiempo de reacción del *jammer* (aplicable a *wormhole*):**
 - **Moverse a un SF menor:** A menor SF, menor es el tiempo en el aire de los mensajes, lo que reduce el margen de tiempo que tiene el *jammer* para activarse (especialmente si es de tipo selectivo). Sin embargo, esto tiene dos consecuencias: menor SF implica menos cobertura y confiabilidad, y se requiere menos energía en el *jammer* para ser efectivo.
 - **Reducir tamaño de mensajes:** Otro resultado de [27] es que el tamaño del mensaje influye significativamente en el tiempo en aire, por lo que reduciendo este (si fuera posible) ayuda a combatir el tiempo de reacción del *jammer*.

- **Conocimiento y control de la tasa de transmisión:** A más alto nivel, si la entidad analizadora del tráfico es consciente de la tasa de transmisión de los ED, es relativamente simple detectar cambios no planificados en los patrones de tráfico y reaccionar acorde a esto.
- **Análisis de DevNonces:** Este es un caso muy particular explicado al final de 3.5.2.5. En [35] se investigó el impacto de *jamming* **exclusivamente** para el control de RSSI en el medio, **no para interferir**, aplicable solo para la generación de DevNonces en los JR. El *hardware* utilizado se basaba en parte del RSSI como “semilla” en el generador pseudo-aleatorio, por lo que se abusó de ello emitiendo un RSSI constante y por medio de análisis de distribuciones con fórmulas particulares de distancia entre DevNonces consecutivos, se detectó que había *jamming* presente influenciando la generación. A pesar de ser un caso tan específico es bueno tenerlo en cuenta para la selección de *hardware*.

5.2.3. Acceso físico

Tanto el acceso físico en sí como la reducción de rango de señal por bloqueos físicos son factores en los que se dificulta dar una recomendación. Es responsabilidad de la organización ubicar los EDs estratégicamente para reducir la probabilidad de robo, bloqueo u acceso indebido. Siempre evitando afectar el rendimiento de los EDs en la medida que sea posible. En las siguientes recomendaciones que impliquen acceso físico, se asume ya explicado este aspecto.

5.2.4. Bypass de mecanismos de protección en *hardware*

La seguridad en *hardware* es un tema complejo. Según [66], los ataques en *hardware* pueden clasificarse en: troyanos de *hardware*, ataques de análisis de canal lateral (*side-channel*), ataques de falla (*fault attacks*), falsificación de chips e ingeniería reversa. Todos ellos merecen una investigación separada y [66] proporciona múltiples referencias sobre ello. Previo a la adquisición de *hardware* es necesario evaluar las medidas de seguridad generales de los productos disponibles, saber si cuentan con partes especializadas integradas como los HSM (*Hardware Security Module*) [67] encargados de realizar operaciones criptográficas y gestionar secretos de forma segura, o si existen mecanismos de verificación de integridad como *secure boot*.

Secure boot es un proceso de verificación del *firmware* realizado en cada encendido para evitar que *firmware* malicioso cargado en el dispositivo sea ejecutado. Según el producto puede variar su implementación, pero su esencia es que un código cargado en una memoria ROM calcule hashes de los binarios y el programa de carga [68], con una clave grabada en memoria OTP (*One Time Programmable*). Particularmente, los chips ESP32 ya mencionados han evolucionado su mecanismo de *secure boot* luego de ser vulnerado por un investigador [14, 15] y además cuentan con otro mecanismo llamado cifrado de *flash* para que los datos de esta memoria no sean legibles. Esta última protección también fue vulnerada pero se mejoraron los chips para mitigar el problema. Esto se plantea como un ejemplo de la importancia de actualizarse e investigar el equipamiento antes de los despliegues o cada cierto tiempo para analizar un recambio.

5.2.5. OTA y circuito integrado de LoRa

Tanto contar con carga inalámbrica de *firmware* (OTA) como tener los circuitos integrados separados entre MCU y transceptor son características particulares de cada producto, especialmente lo segundo que es algo dependiente del diseño y no hay nada que hacer el respecto más que obtener un producto lo más integrado posible. Respecto a OTA, para un proveedor es la opción más práctica ya que permite actualizaciones remotas sin necesidad de acceder dispositivo por dispositivo para cargar el nuevo *firmware*. Sin embargo, esto presenta varios puntos a tener en cuenta [69]:

- **Seguridad en la operativa de despliegue:** Es necesario un servicio de administración de EDs que les informe a los que se encuentren conectados cuándo hay una actualización y cómo obtenerla. Otorgando acceso a este sistema únicamente a usuarios autorizados minimiza la probabilidad de cometer errores o que atacantes inyecten *malware* en los EDs.
- **Despliegue incremental de actualizaciones:** Esto es del lado del proveedor y significa tener control de qué EDs actualizar de forma progresiva para evitar sobrecargas de ser miles de dispositivos y además aplicar parches críticos a los EDs que realmente sea necesario, dado que no necesariamente todos van a estar ejecutando la misma versión de *firmware* simultáneamente.
- **Descarga segura:** En IoT es común que los EDs estén conectados a la nube para intercambiar datos por protocolos como MQTT que pueden montarse sobre un canal seguro con TLS. Una opción es utilizar ese canal, aunque es válido utilizar medios independientes como HTTPS, en tal caso, ampliar las tecnologías disponibles aumenta la superficie de ataque. Si el despliegue es local, sin acceso a Internet, OTA sigue siendo una posibilidad acercándose lo suficiente a los EDs para transferir la actualización. De igual forma, el producto puede soportar utilizar la modulación de LoRa para este fin o una tecnología inalámbrica independiente. Hay que recordar que, en caso de utilizar LoRa, un ataque de *jamming* que afecte la actualización dejaría inconsistente el *firmware* y si el propio ED no puede volver a la versión estable anterior, quedaría inutilizable.
- **Autenticar la descarga:** Es vital que el ED sea capaz de corroborar que la actualización que fue descargada/transferida sea auténtica y su origen es del fabricante. Lo usual es que el fabricante firme la actualización y que el ED cuando la reciba calcule un hash de esta para verificar que las firmas coinciden. En esencia es el mismo procedimiento que en el *secure boot* mencionado en 5.2.4.
- **No interrumpir la operativa:** En el caso de un sensor, que esté inoperativo durante el proceso de actualización puede no significar un riesgo (mientras tarde un tiempo razonable), pero en caso de luces, cerraduras u otro servicio de tiempo real seguramente lo sea. El ED debe ser capaz de actualizar en segundo plano, evitando que un atacante se aproveche de la situación.
- **Revertir la versión:** Como se mencionó en el punto de descarga segura, es importante que el ED pueda volver a la versión anterior del *firmware* en caso de fallas, que podrían darse por muchos motivos como problema de energía o usuario impaciente que lo reinicia. Si el

ED no se recupera, puede quedar en estado no responsivo hasta una carga correcta manual o incluso en estado *bricked* (totalmente inutilizable).

Incluir OTA o no depende de evaluar el cumplimiento de todos estos factores y la realidad del despliegue (no siempre es viable prescindir de actualizaciones remotas o inalámbricas). Si el ED soporta OTA, debería ser posible deshabilitarlo o adquirir una versión del producto que no incluya el componente en el circuito integrado.

5.2.6. Derivar claves de sesión de AppKey

En la activación OTAA, la AppKey es un activo crítico pues es la única clave raíz para generar claves de sesión. Por esto es necesario que toda AppKey sea generada aleatoriamente y única para cada ED para evitar el compromiso múltiple de EDs en caso de que una AppKey sea obtenida. La aleatoriedad vuelve complejo el proceso de deducción ya que pasaría a ser fuerza bruta. En este punto el proceso podría ser acelerado contando con pares de mensajes JR-JA, JR-JR, JA-JA o JA-DataUp para *cracking offline* [22] (aunque no deja de ser fuerza bruta). Si logra obtenerse una AppKey por este medio, el escenario es similar a ABP dado que la AppKey se graba en el ED. Esta situación sólo es evitable cambiando periódicamente la AppKey en los EDs o cambiar a LoRaWAN 1.1, donde introduce a su esquema de seguridad dos claves raíz: AppKey y NwkKey.

En cuanto al robo, el asociado a filtración de datos en el proveedor no es algo que la organización pueda controlar de manera directa, pero al momento de elegir entre diferentes proveedores se debería analizar las garantías y protecciones que ofrecen respecto a la seguridad de los datos. Para el robo en dispositivo el enfoque es mucho más amplio y se trata desglosado en 5.2.7.

5.2.7. Robo de claves en dispositivo

En primer lugar se comenta la rama de acceso remoto administrativo que se basa en que tal acceso está disponible en alguna interfaz de administración y que de lograrlo, las claves se encuentran en alguna sección de configuración. Los nodos correspondientes al subárbol se describen más adelante.

En segundo lugar, el robo de claves en general asume acceso físico y se describieron tres caminos principales. Uno donde las claves se encuentran impresas en el ED (es algo simple de corroborar al haber adquirido los EDs), otro donde las claves se filtran en mensajes de log y otro que requiere un paso más logrando el *bypass* de mecanismos de protección de *hardware*. El segundo es algo común en proyectos fuera del mundo IoT donde puede ser más simple acceder a un log. Si esto se traslada a IoT, hay chances de que los implementadores crean menos probable la lectura de interfaces como una serial y dejen por este motivo (o por error) impresiones de los eventos que ocurren en el ED. Estos eventos pueden ser transmisión/recepción de mensajes, errores internos, o información del proceso de inicio cuando se enciende. En este último caso puede que se impriman datos del ED como DevEUI y claves de sesión, o incluso si es OTAA, cada vez que se da un *join* exitoso. Debe corroborarse que estos registros solo se impriman en situaciones de *debugging* para desarrollo o mantenimiento.

El camino que implica saltar controles en *hardware* se dividió en tres. Si las claves se almacenan en archivos de texto en el ED, lo recomendable es que de alguna forma se cifren para evitar su recuperación. En sí lo ideal es que no figuren en tales fuentes y que se encuentren en una sección protegida de memoria, por ejemplo, gestionada por un HSM como se mencionó en 5.2.4. Tanto para los otros dos casos de ingeniería reversa como análisis de la memoria, se recomienda remitirse al cifrado de la memoria, algo en principio independiente del *secure boot* como mecanismo de protección. No se posee conocimiento suficiente en esta área para ofrecer recomendaciones más específicas ya que además depende del *hardware* en cuestión.

5.2.8. Robo de claves en *network server*

Este caso sigue recomendaciones de protección corrientes para un sistema tradicional (en lo que respecta a su administración y gestión de secretos, **no a su lógica en LoRaWAN**). El NS podría ser gestionado por un sitio web o exponer una API para interactuar con él. No se detallarán mitigaciones para acceso a red interna u obtener credenciales dado que las posibles formas de acceder son varias y no son el enfoque de esta investigación. Particularmente para la existencia de *exploits*, en principio se recomienda periódicamente buscar si existen CVEs (*Common Vulnerabilities and Exposures*) asociados a los sistemas utilizados y mantenerlos actualizados en sus últimas versiones (o al menos que no tengan vulnerabilidades conocidas).

Sí vale la pena comentar que el NS se incluye en esta amenaza porque allí se calculan ambas claves de sesión y es un punto de máxima confianza y de falla a la vez en términos de confidencialidad. En LoRaWAN 1.1 se agregan más servidores al esquema de red y la responsabilidad de generar claves y su gestión es distribuida. De hecho, se introducen cinco claves de sesión diferentes y como ya se comentó en 5.2.6 hay dos claves raíz.

5.2.9. Acceso administrativo al dispositivo

Asumiendo interfaz administrativa disponible (de no existir o estar deshabilitada esta amenaza no tiene lugar), sea remoto o físico, el acceso administrativo requiere poder de acción en el ED. Esto se presentó en los nodos como “Ejecución de comandos” y dependiendo del ED puede ser a través de una API web, comandos a través de protocolo propietario, una terminal con permisos para ejecutar comandos, entre otras. Es necesario que exista algún mecanismo de autenticación para conceder el acceso y utilizar canales cifrados para evitar que las credenciales sean visibles. Asumiendo que todo esto se da, las violaciones de seguridad son más generales como para ofrecer recomendaciones específicas.

El punto que queda es el *exploit* de tecnología inalámbrica. Las tecnologías inalámbricas para gestión suelen ser distintas a las del canal de datos. Por lo que lo más común es utilizar Bluetooth o Wi-Fi para el acceso administrativo. Es importante estar al tanto de las fallas de las tecnologías como los que han surgido para estas dos en particular en lo referente a apareamiento de dispositivos en BLE [3, 11] o la seguridad de WPA2 en Wi-Fi con los ataques KRACK [70]. Nuevamente, la sugerencia es actualizarse basándose en la tecnología y capacidades del *hardware*.

5.2.10. Alterar comportamiento del dispositivo

Lo correspondiente a carga de *firmware* corrupto ya fue tratado en recomendaciones anteriores. En alteración de *hardware*, la única recomendación es restringir lo más posible el acceso físico pues todos los ataques que dependen de ello no tienen contramedidas generales más que esa. Para contramedidas específicas es necesario conocer el *hardware* y, aun así, a priori todos los ataques son viables si las características de este lo permiten.

En lo que respecta a comandos maliciosos por LoRaWAN, el primer paso es verificar si es necesario el uso de *downlink* pues si no lo fuera, deshabilitarlo elimina la amenaza. Sin embargo, bien gestionado el *downlink* es muy útil, especialmente para rescatar a un ED de situaciones como *jamming* indicándole que haga cambios de frecuencia específicos. Lo que resta en ese caso es mejorar la seguridad de las claves de sesión, algo ya cubierto en recomendaciones anteriores.

5.2.11. Replay de mensajes antiguos en la misma sesión

Como ya se explicó en 3.5.2.4, que un atacante envíe mensajes antiguos puede ser un problema distinto según la configuración del NS. Si no se controla el FCnt, cualquier mensaje puede ser repetido y los datos serán válidos luego de procesarse. Si se controla (lo recomendado), puede darse una denegación de servicio si el NS recibe un FCnt mayor al último por parte del atacante, pues todos los mensajes legítimos con FCnt menor serán ignorados, siempre y cuando la diferencia no supere un umbral de $\text{MAX_FCNT_GAP} = 2^{14}$ [71]. Este límite se eliminó en LoRaWAN 1.1 porque no se consideró necesario para FCnt de 32 bits. El mensaje puede ser fabricado por el atacante (no sería *replay* exactamente) o uno ya enviado en la sesión LoRaWAN. Para que esto último tenga efecto, el FCnt debe haber alcanzado su límite y vuelto a 0, momento en el que mensajes anteriores pasan a ser válidos.

En OTAA la solución es comenzar un nuevo *Join Procedure* y renovar las claves de sesión. En ABP no es posible cambiar las claves dinámicamente, por lo que la contramedida es utilizar FCnt de 32 bits siendo que probablemente en la vida útil del ED no se alcance el límite y las soluciones actuales suelen implementarlo. De hecho, en LoRaWAN 1.1 los contadores son obligatoriamente de 32 bits y hay más de dos tipos de contador [72]. Si por restricciones de la realidad fuera obligatorio utilizar 16 bits en ABP, la única sugerencia es actualizar el *firmware* del ED con las nuevas claves de sesión lo antes posible.

Un detalle para tener en cuenta con el control de FCnt, es que si el ED por algún motivo se reinicia en general también vuelve a 0 su contador. Aquí se debe estar alerta porque el NS no aceptará ningún mensaje hasta que se llegue al último FCnt recibido, como gestionar todas estas situaciones dependerá de quien administre la red LoRaWAN.

5.2.12. Bit-flipping en enlace NS-AS

A diferencia del *bit-flipping* entre ED-GW, el enlace NS-AS no es por un canal LoRa, sino con TCP/IP. Por un lado, se requiere capacidad de actuar en el enlace con un ataque MitM lo que ya es complejo, pero si se lograra, la libertad de modificación es completa ya que el control de integridad que ofrecía LoRaWAN termina en el NS. La única integridad que vale es a nivel de

transporte y no de los datos. En conclusión, se debe proteger fuertemente el enlace NS-AS con el método que mejor se ajuste a la realidad de aseguramiento entre estos servidores por parte del administrador de la red.

5.2.13. Exceso diario de mensajes

Esta amenaza es relevante cuando existe regulación automática sobre el flujo de mensajes, respetándose un criterio de *duty cycle* (explicado en 2.3.1 y 3.5.2.5). En Uruguay esto no sucede, pero por completitud se incluye en el análisis.

En OTAA el envío forzoso de JR por bloqueo de sus respectivos JA es un problema importante ya que si se da el caso, probablemente se corresponda a un atacante que posee un *jammer* sofisticado que logra afectar los JA provenientes de los GW o los JR que viajan hacia ellos. Las únicas recomendaciones sobre esto son las ya mencionadas en 5.2.2.

Luego independientemente del modo de activación, si la lógica del ED requiere que lleguen las confirmaciones del NS, el bloqueo de ACKs desde los GWs tiene un efecto similar al caso anterior, pero con el cese en *uplink*. Tratándose de un problema de regulación, se debe evaluar si las confirmaciones de mensajes son necesarias para la realidad y en caso de serlo, remitirse a las contramedidas de *jamming*.

El abuso de comandos MAC respecto a regulación trata de obligar a un ED a contestar mensajes de *downlink* de un atacante construidos con las claves de sesión ya adquiridas. Los ejemplos presentados en 3.5.2.5 son comandos que refieren a evitar que el ED deje de recibir mensajes o transmita en frecuencias no admisibles para el GW o NS, de todas formas, existen diversos tipos de comandos MAC que pueden ser respondidos por un ED en la especificación de LoRaWAN [21]. Algunos ejemplos de comandos son: TxParamSetupReq, RXTimingSetupReq, NewChannelReq, DevStatusReq, RXParamSetupReq, DutyCycleReq y LinkADRRReq. La respuesta del ED es un comando con el mismo nombre, pero en lugar de “Req” al final, lleva “Ans”. La única recomendación para este tipo de problema es monitorear anomalías de comportamiento de los dispositivos.

5.2.14. Replay en Join Procedure

Los *replays* de JR o JA son ataques relativamente viables, especialmente de JA como se explicó en 3.5.2.5. Lamentablemente el control en LoRaWAN 1.0.* no es totalmente efectivo y las mejoras se dieron en la versión 1.1, donde el *Join Procedure* incrementó su complejidad. Las contramedidas para los *replays* en OTAA son:

- Monitorear el tráfico con énfasis en los *Join Procedures*. Si se detectan anomalías en el flujo de JRs en el GW o NS, sospechar, analizar y si se cree necesario, cambiar la AppKey en el ED afectado dado que con ella se firman los JRs y los mensajes que el atacante almacenó ya no serán válidos.
- Actualizar a LoRaWAN 1.1, un cambio sustancial que trae consigo sus propias nuevas vulnerabilidades, pero fue diseñado con las mejoras en el *Join Procedure* en mente, entre otras cosas.

- Cambiar a activación ABP no es la mejor recomendación, siendo que no es el método preferido por sus problemas de seguridad ya planteados. Aunque particularmente en este escenario, las amenazas de *replay* de JRs y JAs dejarían de existir.

6. Conclusiones y trabajo futuro

Fue un proyecto desafiante. Existió esfuerzo adicional dado que, a pesar de que se contaba con conocimiento en distintas áreas integradas en el proyecto (seguridad, redes y arquitectura de computadoras), nunca se había practicado el modelado de amenazas ni la interacción con dispositivos a bajo nivel, no se conocía LoRaWAN, ni se había trabajado con radiofrecuencia. Tomó tiempo significativo aprender sobre todas estas cuestiones a lo largo del proyecto. IoT es un área verde con actual investigación en la universidad, pero en general en la literatura no se cuenta con tanto material bibliográfico (especialmente de seguridad) comparando con otras tecnologías. Afortunadamente, en noviembre de 2019 se participó del Workshop de Sistemas Ciberfísicos (WSCF)²³ en Facultad de Ingeniería, particularmente en el taller “IoT hands-on tour” donde se aprendió sobre el trabajo con sensores utilizando los SparkX de este mismo proyecto.

Una de las grandes dificultades sobre la que no se tuvo control fue la pandemia desde marzo de 2020, factor que impidió tener reuniones para trabajar con el *hardware*, hacer pruebas en el exterior e incluso conseguir equipamiento. Esto último se comenta porque no todos los comercios de electrónica estaban disponibles e incluso el que tenía un producto necesario agotó su *stock* por un cliente que compró todas las unidades disponibles el día que la tienda volvió a funcionar. Para pruebas a nivel electrónico, el escenario ideal habría sido asesorarse con expertos de facultad en persona, pero esto no se consideró viable y el trabajo se llevó a cabo cuidadosamente (siguiendo advertencias para evitar dañar el *hardware*), invirtiendo quizás más tiempo del esperado.

En otras cuestiones técnicas también se identificó la limitación de los SparkX por ser dispositivos de un canal, a diferencia de The Things Gateway que fue una pieza fundamental en el proyecto, aunque al ser tecnología propietaria no fue tan personalizable como se hubiera querido. Esto no quita que los SparkX fueran clave, su programabilidad fue lo que permitió construir el *jammer* en primer lugar. Por otra parte, el trabajo con radiofrecuencia en general fue dificultoso, no siempre podía observarse la presencia de tráfico. Un caso claro fue implementando el *jammer*, pues no era posible ver si se estaba emitiendo realmente y se deducía únicamente por la ausencia de mensajes legítimos. Otro caso fue detectar *Join-Requests*, que por motivo que se desconoce no podían verse, pero sí sus *Join-Accepts* asociados. Esto fue prohibitivo para probar ataques como *replay* o envío forzoso de JR, siendo que es necesario detectarlos²⁴.

Sumando las limitaciones de la pandemia a las del *hardware*, los resultados de las pruebas no reflejan un escenario con la escala de ciudades inteligentes, sino mucho más reducido, suficiente para demostrar empíricamente lo que la teoría expresa y sacar conclusiones extendiendo el razonamiento a proporciones de ciudad. Entonces a continuación, se resumen los resultados del proyecto.

Desde el punto de vista del consumidor, IoT es un paradigma que ofrece miles de oportunidades de desarrollo tecnológico en múltiples ámbitos, desde hogar y salud hasta ciudades e industrias. Seguir explorando las tecnologías IoT es el camino para mejorar el rendimiento y calidad en

²³Disponible en: <https://www.fing.edu.uy/inco/grupos/mina/wscf2019/>, último acceso: 7 de julio de 2020.

²⁴Solo en una ocasión se pudo visualizar un JR en la banda US, no volvió a ocurrir. Se estima que hubo algún inconveniente técnico o de configuración en el sitio de TTN o su GW.

los procesos de negocio a los que aplica. Particularmente, la tecnología LoRaWAN es una muy buena solución en el ámbito ciudades inteligentes e industrias que exijan extensa cobertura de red, como la ganadera. Una buena implementación LoRaWAN puede cubrir kilómetros de comunicación con escaso consumo energético por su diseño de LPWAN.

Desde el punto de vista del atacante, la seguridad IoT es un terreno poco explorado, lo que implica en general poco control o conocimiento en medidas de seguridad de las implementaciones. En tecnologías emergentes es fácil creer que se está seguro por falta de experiencia en el campo incluso para los atacantes o se confía ciegamente en los fabricantes y diseñadores asumiendo que hoy en día la seguridad es parte del producto. Afortunadamente, LoRaWAN tuvo foco en la seguridad desde el diseño, aunque este trabajo se encargó de recopilar evidencia y confirmar que de todas formas existen vulnerabilidades en la tecnología.

Así como se avanza velozmente los nuevos desarrollos, también se evoluciona en las mejoras y es por esto que surgió LoRaWAN 1.1, rediseñando el protocolo para contemplar los errores de seguridad identificados en la literatura. Sin embargo, no está libre de problemas de seguridad por características de su naturaleza. Por supuesto, esto no implica que LoRaWAN no pueda implementarse. Al planificar el despliegue deben contemplarse todas las vulnerabilidades, sus contramedidas y en base a esto adquirir equipamiento y sistemas que sean robustos y seguros. La infraestructura debe ser capaz de **monitorearse** y es vital tener capacidad de reacción y control sobre tantos dispositivos, identificando patrones de comunicación para detectar anomalías. Una buena gestión de actualizaciones es importante ante eventuales fallas de *firmware* o necesidad de cambio de claves (de sesión en caso de ABP o AppKey en OTAA). Todas estas medidas permiten rápida acción ante ataques inevitables como el *jamming* de señal, dado que el aire es un medio compartido y se carece de control sobre el uso de este. De hecho, en instancias de pruebas se llegaron a detectar mensajes de múltiples dispositivos no identificados.

Además, se destaca la importancia de la protección contemplando que, el equipamiento utilizado en este proyecto o similar alcanza para lograr impacto y su costo es relativamente bajo. SparkFun ofrece un ESP32 (mejora del SparkX) a 35 dólares en su sitio oficial²⁵ y con algunos de ellos ya es posible construir un laboratorio de pruebas y ataque. Otros dispositivos más costosos como The Things Gateway que ronda los 380 dólares²⁶ son una inversión, con ellos es posible abarcar en general ocho canales simultáneamente y si el *firmware* del dispositivo es modificable, se abren diversas puertas para implementar ataques. Todo esto sin considerar que lo implementado en este proyecto está basado en código abierto y gratuito y que seguramente se pueda conseguir otro tipo de *hardware* menos costoso.

A nivel de *hardware*, se aprendió que es imprescindible la presencia de mecanismos de protección que verifiquen integridad del *firmware* y que almacenen y gestionen de forma segura los secretos que se utilizan. Además, es deseable que los dispositivos soporten los comandos MAC de LoRa asociados a cambios en parámetros de transmisión, por ejemplo, el cambio de frecuencias a pe-

²⁵Disponible en: <https://www.sparkfun.com/products/15006>, último acceso: 7 de julio de 2020.

²⁶Disponible en: <https://www.newark.com/the-things-network/ttn-gw-915/accessory-type-wireless-gateway/dp/05AC1807>, último acceso: 7 de julio de 2020.

dido del administrador de la red. Contemplando estas y todas las recomendaciones de 5.2 un despliegue LoRaWAN (incluso en su versión 1.0.3) reduce drásticamente su superficie de exposición y probabilidad de éxito en los ataques identificados.

Otro aprendizaje es la utilidad del modelado de amenazas para identificar la raíz de los ataques que surgen de este. Por otro lado, el modelo resulta complejo y ramificado al punto de que es necesario acotar el alcance. De todas formas, dicho alcance puede ser suficiente para abarcar diversos ataques que indirectamente involucran a entidades que fueron excluidas inicialmente (como sucedió con el GW y NS habiéndose centrado en el ED). De hecho, el enfoque con DFD, STRIDE y árboles de ataque fue la base de este trabajo. Aquí es pertinente mencionar la herramienta SHARPE como elemento adicional de comprobado aporte en la medición de relevancia de nodos en un árbol de ataque y su análisis probabilístico en general.

Antes de lo referente a resultados generados como contenido nuevo, en primer lugar, se comenta que el alcance inicial fue poco claro. Se estimaba que el estudio luego del estado del arte derivaría en otros enfoques de IoT como el estudio de seguridad de la plataforma FIWARE o los protocolos de datos creados considerando las restricciones de los dispositivos. Luego entre la necesidad de la Intendencia de Montevideo por investigar LoRaWAN y descubrir que había relativamente poco material en seguridad, aunque suficiente como para investigar, se aventuró a explorar las superficies de ataque con una motivación un tanto ofensiva (desde la visión de un atacante).

A la fecha, no existe en la literatura una investigación que contenga los siguientes elementos descritos en este trabajo: diagramas de flujo de datos con los elementos de LoRaWAN, identificación de amenazas con el modelo STRIDE, árboles de ataque que agrupen amenazas y ataques conocidos (con análisis probabilístico, aunque haya sido parcial para un árbol), y la construcción documentada con prueba de concepto de un *jammer* LoRa. Se cree que todo esto contribuye localmente al instituto de computación y globalmente a la comunidad, como un trabajo extensible y facilitador para las diversas ramas que se mencionan más adelante para trabajo futuro.

A modo de autocrítica, se quiso explorar ataques específicos a la activación OTAA, pero además de las dificultades ya mencionadas, la gestión de tiempo en algunos casos fue deficiente, especialmente investigando posibles soluciones para la experimentación (librerías, *frameworks* y demás). En algunas de estas situaciones el tiempo se consideró desperdiciado en términos de la investigación porque esos resultados no se incluyeron. Un ejemplo claro de esto fue un *dongle* LoRa USB²⁷ (Figura 36) que se investigó por al menos una semana al inicio del proyecto y con el conocimiento del momento no se consiguió hacerlo funcionar para futuras pruebas. Quizás con el conocimiento actual se pueda entender su funcionamiento, pero ya en la etapa cierre de la investigación no había lugar para incluirlo y por eso no figura en el equipamiento utilizado.

²⁷Disponible en: <http://www.ifroglab.com/en/?p=6536>, último acceso: 7 de julio de 2020.



Figura 36: *Dongle* LoRa USB de iFrogLab.

Imagen del sitio oficial: <http://www.ifroglab.com/en/?p=6536>

En lo que respecta a trabajo futuro, hay distintas ramas para comenzar y otras para profundizar:

- El modelo de amenazas generado excluyó al AS y parcialmente al NS por las razones detalladas al final de la introducción de 3.3. No obstante, la gestión de activos en ellos es crucial y partiendo de sus DFD correspondientes podría extenderse el modelo para completar el análisis.
- También la investigación podría ampliarse mirando hacia los integradores de datos y las plataformas IoT como FIWARE, que son esenciales en el tratamiento de los datos y administración IoT en general. Si se cree de interés, podría hacerse un modelado de amenazas sobre alguna plataforma en particular contemplando todos los sistemas con los que potencialmente se integra.
- El GW se vio involucrado fuertemente en el modelo de amenazas generado, aunque no fue centro de estudio y podrían haber amenazas que surjan de su investigación si se profundiza [73].
- Los ataques llevados a cabo no alcanzan a cubrir el espacio de ataques identificados, principalmente por probabilidad de éxito, falta de experiencia en temas específicos o limitaciones del equipamiento. Investigaciones futuras con *hardware* más potente, distintas condiciones de prueba y tiempo ahorrado basándose en resultados de esta investigación, seguramente obtengan nuevos o mejores resultados. Especialmente en las pruebas que no tuvieron éxito, como el *jammer* reactivo. Particularmente en este último ejemplo, también se dio una restricción a nivel del *software* actual que no pudo sobrellevarse. Se insta a involucrar experiencia de las áreas de electrónica y señales como apoyo para los trabajos por venir.
- Los EDs de este trabajo se consideraron básicos. Es buena opción investigar sobre la seguridad en capacidades de EDs que reciban comandos por *downlink* y además oficien de “actuadores” (popularmente son el opuesto a los sensores que son pasivos). El resultado es un impacto directo en acciones que estos ejercen en su ambiente físico. Un ejemplo que

involucra ambos dispositivos es un sensor de movimiento que detecta una presencia indebida y lo comunica a un actuador que automáticamente activa una cerradura mecánica. Lo interesante en este campo es estudiar las amenazas relativas a la reacción de un actuador.

- En lo que refiere a capacidades de LoRaWAN no tratadas en este trabajo, se incluyen *roaming* para movilidad de EDs, mensajes *multicast* y profundización en *downlink* en general. La información base se encuentra en la especificación de LoRaWAN [21], aunque no es tan completa y lo mejor es recurrir a trabajos que hayan estudiado estos aspectos.
- Uno de los puntos más importantes de investigación a futuro en LoRaWAN es la versión 1.1. En ella se hicieron cambios en el esquema general sumando servidores, claves raíz, claves de sesión, contadores y varios elementos extra [72]. Afortunadamente ya hay investigaciones que evalúan LoRaWAN 1.1. Las dos que se destacan son un análisis de compatibilidad hacia atrás con LoRaWAN 1.0.2 enfocado en la seguridad [34] y un muy completo análisis de las amenazas en LoRaWAN 1.1 repasando las de versiones anteriores [74]. Se recomienda fuertemente acudir a este último trabajo.
- Relacionado al punto anterior, se anunció que en el correr de 2019 se publicaría LoRaWAN 1.0.4 y en 2020 la 1.1.1 [75]. Al momento de escribir esto (7/7/2020) no fueron publicadas, por esto no se ha hecho ninguna recomendación o adelanto que las involucre. Desde Semtech, la empresa creadora de la tecnología, se recomienda utilizar la versión 1.0.2 a menos que se precisen EDs clase B, en cuyo caso se recomienda la 1.0.3. Aunque adelantaron que 1.0.4 será la versión de preferencia cuando se publique.
- Este punto no es sobre LoRaWAN, pero otra rama de estudio es la seguridad en los protocolos de datos. En 2.1 se listan los destacados popularmente y existen investigaciones sobre su desarrollo.
- Para continuar evaluando los árboles de ataque contruidos, se sugiere profundizar el análisis probabilístico con SHARPE en los modelos ya creados para trabajos futuros. Como las probabilidades son ajustables a cada realidad, puede observarse la probabilidad final para los árboles antes y después de las contramedidas, cuantificando la reducción de riesgo.

Concluyendo este trabajo, se invita a la constante actualización en las cuestiones de seguridad de tecnologías emergentes, dado que si no se va al paso de la tecnología no solo alguien más encontrará los problemas que uno no ve, sino que se podría llegar al escenario planteado por el escritor Arthur C. Clarke:

“Cualquier tecnología suficientemente avanzada es indistinguible de la magia”

Referencias

- [1] Saleh Soltan, Prateek Mittal, and H. Vincent Poor. BlackIoT: IoT Botnet of High Wattage Devices Can Disrupt the Power Grid. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC’18, page 15–32, USA, 2018. USENIX Association.
- [2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, August 2017. USENIX Association.
- [3] Mike Ryan. Bluetooth: With Low Energy Comes Low Security. In *7th USENIX Workshop on Offensive Technologies (WOOT 13)*, Washington, D.C., August 2013. USENIX Association.
- [4] LoRa® Alliance Technical Marketing Workgroup. What is LoRaWAN®. A technical overview of LoRa® and LoRaWAN™ Technical.
<https://loro-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf>
[Accedido: 7/7/2020].
- [5] Microsoft. The STRIDE Threat Model.
[https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878\(v%3dcs.20\)](https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v%3dcs.20)) [Accedido: 7/7/2020].
- [6] D. Bastos, M. Shackleton, and F. El-Moussa. Internet of things: A survey of technologies and security risks in smart home and city environments. In *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, pages 1–7, 2018.
- [7] Gartner. Gartner Says 8.4 Billion Connected “Things” Will Be in Use in 2017, Up 31 Percent From 2016, February 2017.
<https://www.gartner.com/newsroom/id/3598917> [Accedido: 7/7/2020].
- [8] C. Thomás Oliveira, R. Moreira, F. de Oliveira Silva, R. Sanches Miani, and P. Frosi Rosa. Improving Security on IoT Applications Based on the FIWARE Platform. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 686–693, 2018.
- [9] Ruth Ande, Bamidele Adebisi, Mohammad Hammoudeh, and Jibran Saleem. Internet of things: Evolution and technologies from a security perspective. *Sustainable Cities and Society*, 54:101728, 2020.
- [10] Shodan. The world’s first search engine for Internet-connected devices.
<https://www.shodan.io/> [Accedido: 7/7/2020].
- [11] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B. Rasmussen. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1047–1061, Santa Clara, CA, August 2019. USENIX Association.

- [12] Espressif. ESP32 Series Datasheet.
https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf [Accedido: 7/7/2020].
- [13] SparkFun. Sparkfun SparkX ESP32 LoRa 1-Channel Gateway (Model: SPX-14893).
<https://www.sparkfun.com/products/retired/14893> [Accedido: 7/7/2020].
- [14] LimitedResults. Pwn the ESP32 Forever: Flash Encryption and Sec. Boot Keys Extraction, November 2019.
<https://limitedresults.com/2019/11/pwn-the-esp32-forever-flash-encryption-and-sec-boot-keys-extraction/> [Accedido: 7/7/2020].
- [15] LimitedResults. Pwn the ESP32 Secure Boot, September 2019.
<https://limitedresults.com/2019/09/pwn-the-esp32-secure-boot/> [Accedido: 7/7/2020].
- [16] Matheus Garbelini. Proof of Concept of ESP32/8266 Wi-Fi vulnerabilities (CVE-2019-12586, CVE-2019-12587, CVE-2019-12588) . Repositorio de GitHub.
https://github.com/Matheus-Garbelini/esp32_esp8266_attacks [Accedido: 7/7/2020].
- [17] OWASP Foundation. OWASP Internet of Things Project.
<https://owasp.org/www-project-internet-of-things/> [Accedido: 7/7/2020].
- [18] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne. Understanding the Limits of LoRaWAN. volume 55, pages 34–40, 2017.
- [19] Raphael Rom and Moshe Sidi. Aloha Protocols. In *Multiple Access Protocols: Performance and Analysis*, pages 47–77, New York, NY, 1990. Springer New York.
- [20] Gemalto, Actility and Semtech. LoRaWANTM Security, full end-to-end encryption for IoT application providers.
https://lora-alliance.org/sites/default/files/2019-05/lorawan_security_whitepaper.pdf [Accedido: 7/7/2020].
- [21] LoRa AllianceTM. LoRaWAN[®] Specification v1.0.3, July 2018.
<https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf> [Accedido: 7/7/2020].
- [22] Cesar Cerrudo, Esteban Martinez Fayo, Matias Sequeira. LoRaWAN Networks Susceptible to Hacking: Common Cyber Security Problems, How to Detect and Prevent Them, January 2020.
<https://act-on.ioactive.com/acton/attachment/34793/f-87b45f5f-f181-44fc-82a8-8e53c501dc4e/1/-/-/-/-/LoRaWAN%20Networks%20Susceptible%20to%20Hacking.pdf> [Accedido: 7/7/2020].
- [23] S. Tomasin, S. Zulian, and L. Vangelista. Security Analysis of LoRaWAN Join Procedure for Internet of Things Networks. In *2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 1–6, 2017.

- [24] OWASP Foundation. Threat Modeling - OWASP Cheat Sheet Series.
https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html [Accedido: 7/7/2020].
- [25] OWASP Foundation. OWASP Threat Model Cookbook.
<https://owasp.org/www-project-threat-model-cookbook/> [Accedido: 7/7/2020].
- [26] Inger Anne Tøndel, Martin Gilje Jaatun, and Maria Bartnes Line. Threat Modeling of AMI. In Bernhard M. Hämmerli, Nils Kalstad Svendsen, and Javier Lopez, editors, *Critical Information Infrastructures Security*, pages 264–275, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [27] Emekcan Aras, Nicolas Small, Gowri Sankar Ramachandran, Stéphane Delbruel, Wouter Joosen, and Danny Hughes. Selective Jamming of LoRaWAN Using Commodity Hardware. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MobiQuitous 2017, page 363–372, New York, NY, USA, 2017. Association for Computing Machinery.
- [28] M. G. Jaatun and I. A. Tøndel. Covering Your Assets in Software Engineering. In *2008 Third International Conference on Availability, Reliability and Security*, pages 1172–1179, 2008.
- [29] Robert Miller, MWR Labs. LoRa Security: Building a secure LoRa solution, March 2016.
<https://labs.f-secure.com/assets/BlogFiles/mwri-LoRa-security-guide-1.2-2016-03-22.pdf> [Accedido: 7/7/2020].
- [30] JungWoon Lee, DongYeop Hwang, JiHong Park, and Ki-Hyung Kim. Risk analysis and countermeasure for bit-flipping attack in LoRaWAN. In *2017 International Conference on Information Networking (ICOIN)*, pages 549–551, 2017.
- [31] C. Huang, C. Lin, R. Cheng, S. J. Yang, and S. Sheu. Experimental Evaluation of Jamming Threat in LoRaWAN. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pages 1–6, 2019.
- [32] Sheila A. Berta. Backdooring Hardware Devices By Injecting Malicious Payloads On Microcontrollers. Black Hat USA, August 2019.
<https://www.blackhat.com/html/webcast/08222019-backdooring-hardware-devices-by-injecting-malicious-payloads-on-microcontrollers.html> [Accedido: 9/7/2020].
- [33] Dirección Nacional de Comunicaciones. Resolución 35 2000, Exp. 9903040/1/1802, Reglamento de instalación y operación de sistemas inalámbricos de transmisión de datos en las bandas de frecuencias 902-928 MHz., 2,4-2,4835 GHz. y 5,725-5,850 GHz., 28/01/2000.
- [34] Tahsin C.M. Dönmez and Ethiopia Nigussie. Security of LoRaWAN v1.1 in Backward Compatibility Scenarios. volume 134, pages 51–58, 2018. The 15th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2018) / The 13th International Conference on Future Networks and Communications (FNC-2018) / Affiliated Workshops.

- [35] S. M. Danish, A. Nasir, H. K. Qureshi, A. B. Ashfaq, S. Mumtaz, and J. Rodriguez. Network Intrusion Detection System for Jamming Attack in LoRaWAN Join Procedure. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, 2018.
- [36] C. Hirel, R. Sahner, X. Zang, and K. Trivedi. Reliability and Performability Modeling Using SHARPE 2000. In Boudewijn R. Haverkort, Henrik C. Bohnenkamp, and Connie U. Smith, editors, *Computer Performance Evaluation. Modelling Techniques and Tools*, pages 345–349, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
- [37] Duke University. SHARPE Portal.
<https://sharpe.pratt.duke.edu/> [Accedido: 15/8/2020].
- [38] CERTuy. Estadísticas de incidentes del primer semestre de 2018, August 2018.
<https://www.gub.uy/centro-nacional-respuesta-incidentes-seguridad-informatica/datos-y-estadisticas/estadisticas/estadisticas-de-incidentes-del-primer-semestre-de-2018> [Accedido: 7/7/2020].
- [39] Verizon. 2020 Data Breach Investigations Report.
<https://enterprise.verizon.com/resources/reports/dbir/2020/introduction/> [Accedido: 18/6/2020].
- [40] HOPERF. RFM95/96/97/98(W) - Low Power Long Range Transceiver Module.
https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/RFM95_96_97_98W.pdf [Accedido: 7/7/2020].
- [41] Whitecat. Lua RTOS ESP32. Repositorio de GitHub.
<https://github.com/whitecatboard/Lua-RTOS-ESP32> [Accedido: 7/7/2020].
- [42] SparkFun. Tutorial: ESP32 LoRa 1-CH Gateway, LoRaWAN, and the Things Network.
<https://learn.sparkfun.com/tutorials/esp32-lora-1-ch-gateway-lorawan-and-the-things-network/all> [Accedido: 7/7/2020].
- [43] THINGS4U. Version 6 of the single channel gateway. Repositorio de GitHub.
<https://github.com/things4u/ESP-1ch-Gateway> [Accedido: 7/7/2020].
- [44] The Things Network. The things gateway.
<https://www.thethingsnetwork.org/docs/gateways/gateway/> [Accedido: 7/7/2020].
- [45] The Things Network. Console.
<https://console.thethingsnetwork.org/> [Accedido: 7/7/2020].
- [46] The Things Network. Gateway activation.
<https://activate.thethingsnetwork.org/> [Accedido: 7/7/2020].
- [47] The Things Network. The Things Node.
<https://www.thethingsnetwork.org/docs/devices/node/> [Accedido: 7/7/2020].
- [48] The Things Network. The Things Node: Arduino Setup.
<https://www.thethingsnetwork.org/docs/devices/node/quick-start.html> [Accedido: 7/7/2020].

- [49] SparkFun Electronics. SparkFun Arduino Boards Setup. Repositorio de GitHub.
https://github.com/sparkfun/Arduino_Boards#installation-instructions
[Accedido: 7/7/2020].
- [50] The Things Network. LoRaWAN Frequency Plans and Regulations by Country.
<https://www.thethingsnetwork.org/docs/lorawan/frequencies-by-country.html>
[Accedido: 7/7/2020].
- [51] The Things Network. Frequency plans.
<https://www.thethingsnetwork.org/docs/lorawan/frequency-plans.html>
[Accedido: 7/7/2020].
- [52] IOActive. LoRaWAN Auditing Framework. Repositorio de GitHub.
<https://github.com/IOActive/laf> [Accedido: 7/7/2020].
- [53] Docker. Docker compose.
<https://docs.docker.com/compose/> [Accedido: 7/7/2020].
- [54] Mauro Riva. ESP32 using MicroPython meets LoRa and LoRaWAN. Repositorio de GitHub.
<https://github.com/lemariva/uPyLoRaWAN> [Accedido: 7/7/2020].
- [55] Matthijs Kooijman Thomas Telkamp. Arduino - LMIC library. Repositorio de GitHub.
<https://github.com/matthijskooijman/arduino-lmic> [Accedido: 7/7/2020].
- [56] Sakshama Ghosly. LoRa: Orthogonality.
https://www.sghosly.com/p/lor_a_6.html [Accedido: 7/7/2020].
- [57] Vitor Fialho and Fernando Azevedo. Wireless Communication Based on Chirp Signals for LoRa IoT Devices. *i-ETC: ISEL Academic Journal of Electronics Telecommunications and Computers*, 4(1):6, 2018.
- [58] Lobar. C-Lang LoRaWAN packet parser & encoder. Repositorio de GitHub.
<https://github.com/lobaro/util-lorawan-packets> [Accedido: 7/7/2020].
- [59] Elasticsearch - The heart of the free and open Elastic Stack.
<https://www.elastic.co/elasticsearch/> [Accedido: 9/7/2020].
- [60] InfluxDB - A time series database designed to handle high write and query loads.
<https://www.influxdata.com/products/influxdb-overview/> [Accedido: 9/7/2020].
- [61] Graylog - Gain visibility into your security.
<https://www.graylog.org/> [Accedido: 9/7/2020].
- [62] Matias Sequeira. LoRaWAN Security Research Paper is now published!, The Things Network Forum.
<https://www.thethingsnetwork.org/forum/t/lorawan-security-research-paper-is-now-published/33753> [Accedido: 9/7/2020].
- [63] Paul Hayes, Alliot Technologies. LoRaWAN Security, February 2020.
<https://www.alliot.co.uk/lorawan-security/> [Accedido: 9/7/2020].

- [64] Alejandro Lampropulos. How to build a Jammer Detector, April 2018.
<https://medium.com/worldsensing-techblog/how-to-build-a-jammer-detector-d4e3dd919faa> [Accedido: 7/7/2020].
- [65] Max Ingham, Jims Marchang, and Deepayan Bhowmik. IoT Security Vulnerabilities and Predictive Signal Jamming Attack Analysis in LoRaWAN. January 2020.
- [66] J. Dofe, J. Frey, and Q. Yu. Hardware security assurance in emerging IoT applications. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2050–2053, 2016.
- [67] Dirk Rosenau, FP Inovolabs. The Benefits of a Hardware Security Module in Industrial IoT Applications, 2018.
https://cdn0.scrvt.com/fa130d14754799caab3c404eab497d13/a39a6e45ec63ba9e/b6b0995668f3/FP_Whitepaper_Hardware-Security-Module_IoT40_EN_11052018.pdf [Accedido: 7/7/2020].
- [68] INVIA. Implementing Secure Boot in Your Next Design, 2019.
<https://www.invia.fr/Tutorials/Implementing%20Secure%20Boot%20in%20Your%20System%20on%20Chip%20or%20FPGA%201.0.pdf> [Accedido: 7/7/2020].
- [69] Nick Lethaby, Texas Instruments. A more secure and reliable OTA update architecture for IoT devices, October 2018.
<https://www.ti.com/lit/wp/sway021/sway021.pdf> [Accedido: 7/7/2020].
- [70] Mathy Vanhoef and Frank Piessens. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. October 2017.
- [71] LoRa Alliance™. LoRaWAN™ Regional Parameters, July 2016.
https://lora-alliance.org/sites/default/files/2018-05/lorawan_regional_parameters_v1_0-20161012_1397_1.pdf [Accedido: 7/7/2020].
- [72] LoRa Alliance™. LoRaWAN® Specification v1.1, October 2017.
https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_v1.1.pdf [Accedido: 20/6/2020].
- [73] Lukas Simon Laufenberg. Impersonating LoRaWAN gateways using Semtech Packet Forwarder. volume abs/1904.10728, March 2019.
- [74] Ismail Butun, Nuno Pereira, and Mikael Gidlund. Security Risk Analysis of LoRaWAN and Future Directions. volume 11, page 3, 12 2018.
- [75] Derek Hunt, Semtech. Selecting a LoRaWAN® Specification, Semtech Tech Journal, May 2019.
<https://tech-journal.semtech.com/selecting-a-lorawan-specification> [Accedido: 7/7/2020].
- [76] Barani Design. LoRaWAN USA frequencies, channels and sub-bands for IoT device.
<https://www.baranidesign.com/faq-articles/2019/4/23/lorawan-usa-frequencies-channels-and-sub-bands-for-iot-devices> [Accedido: 7/7/2020].

- [77] Thomas Laurenson. Dragino LoRaShield - Node Configuration for Australian and New Zealand Frequencies (AU915 MHz).
<https://www.thomaslaurenson.com/blog/2018/07/06/dragino-lorashield-node-configuration-for-AU915/> [Accedido: 7/7/2020].
- [78] Microchip. RN2903.
<https://www.microchip.com/wwwproducts/en/RN2903> [Accedido: 7/7/2020].
- [79] The Things Network Forum. HowTo: Update RN2483 Firmware on The Things Node.
<https://www.thethingsnetwork.org/forum/t/howto-update-rn2483-firmware-on-the-things-node/11777> [Accedido: 7/7/2020].
- [80] Microchip. RN2903 SA AU915 v1.0.3.
http://ww1.microchip.com/downloads/en/DeviceDoc/RN2903_SA1.0.3.zip
[Accedido: 7/7/2020].
- [81] Microchip. LoRa(R) Technology Evaluation Kit - 800.
<https://www.microchip.com/Developmenttools/ProductDetails/dv164140-1>
[Accedido: 7/7/2020].
- [82] SparkFun. Board definitions for SparkFun-manufactured AVR, ARM, and ESP-based Arduino boards. Repositorio de GitHub.
https://github.com/sparkfun/Arduino_Boards/tree/master/sparkfun/avr/signed_driver [Accedido: 17/6/2020].

A. Actores relevantes en modelado de amenazas

<i>Actor</i>	<i>Descripción</i>		<i>Objetivos</i>	<i>Probabilidad</i>	<i>Prioridad</i>
Script kiddie	Persona que utiliza ejecutables o código existentes para comprometer objetivos, carente de la experiencia para escribir los suyos.		Lograr acceso a dispositivos IoT o a la infraestructura de la organización.	Alta	1
Hacker especializado	Hacker que es contratado por algún otro actor, que aprovecha de su experiencia para comprometer a un objetivo.		Obtener información sobre los activos, datos de usuarios o cualquier tipo de información que pueda utilizar para ganar mayor acceso y sacarle provecho a la organización. Puede además causar daños/disfunciones en la infraestructura.	Alta	1
Empleado	Empleado de alguna de las partes con acceso al despliegue y busca sacar provecho propio, se consideran como los más relevantes:	Administrador de servicios e infraestructura LoRa	Utilizar los servicios o datos de manera no esperada, abusando de las funcionalidades del rol de empleado, para causar algún tipo de daño a la organización o extraer información que le sea útil.	Alta	2
		Administrador de infraestructura IoT		Media	
		Seguridad de la información		Media	
Competencia	Empresa competidora de la organización o de un proveedor de esta.		Perjudicar a la organización con daños de imagen, económicos, etc. Quitar del camino a un proveedor que es competencia.	Media	3
Ex empleado	Persona que conoce la infraestructura, servicios y detalles del despliegue o información de usuarios.		Afectar la imagen de la organización y buscar provecho económico por la divulgación de información.	Media / baja	4
Entes de control	Organismos de control.		Sancionar a la organización por el incumplimiento de normativas en el uso del protocolo LoRa.	Media	5

Tabla 6: Actores relevantes en las potenciales amenazas.

B. Modificaciones a The Things Node

Aquí se explican las modificaciones necesarias a The Things Node (*firmware* y librerías) para facilitar las pruebas con un único canal y cambiar su frecuencia entre las bandas US y AU. Ubicando la carpeta de la librería de TTN para Arduino (varía según el sistema operativo y ruta de *sketchbook*), se identificó que en las funciones de configuración de canales `configureUS915` y `configureAU915` del archivo `TheThingsNode.cpp` podía cambiarse la condición que decide qué canales activar. En estas funciones (son idénticas en lógica) se debe cambiar la lógica del siguiente `if`:

```
if (ch == ch500 || (ch <= chHigh && ch >= chLow))
```

por:

```
if (ch == 8)
```

El canal 8 es un ejemplo, pero podría utilizarse cualquier canal válido (para referencias sobre la relación canal/frecuencia según US o AU ver [76, 77]). Luego de este cambio, el código cargado en TTNode utilizará una única frecuencia para su envío de mensajes.

El dispositivo The Things Node (TTNode) posee el módulo de LoRaWAN Modulab RN2093 [78]. Por defecto el *firmware* con el que viene está en la versión 0.9.5 y funciona para la frecuencia de US. Pero es posible actualizarlo para utilizarlo en la frecuencia de AU, siguiendo una guía que se encuentra en el foro de TTN [79], si bien la guía es para el chip RN2483, también aplica para RN2903.

- Primero se debe descargar el *firmware* desde la web de soporte de Microchip [80] que corresponde a la versión: **RN2903 SA AU915 v1.0.3**.
- En segundo lugar, es necesario conectar el dispositivo TTNode a un PC y mediante el IDE de Arduino (asumiendo que ya se tienen cargadas las librerías de TTN) ir a: File → Examples → TheThingsNode → Passthrough, y cargar este *sketch* en el dispositivo.
- Por último, para actualizar el *firmware* es necesario descargar la herramienta “LoRa Development Utility” de Microchip [81]. A pesar de que está disponible en varios sistemas operativos, se intentó usar la versión de MacOS y no inició, por lo que se optó por usar la versión de Windows, la cual funcionó correctamente.
- Una vez instalada la herramienta con sus dependencias correspondientes (Java), se debe iniciar el *software* y conectar el TTNode. Una vez que sea identificado, se accede a la pestaña DFU y se selecciona el archivo hexadecimal del *firmware* descargado anteriormente en la sección “bootloader input”. Este archivo se encuentra dentro del ZIP descargado, en la carpeta `binary\For Bootloader`, luego hacer click en “Update Firmware” y esperar a que se complete.
- Una particularidad de conectar el TTNode en Windows, es que puede suceder que no se

identifiquen/instalen los *drivers* adecuados para que sea detectado. En este caso se pueden descargar desde este repositorio de GitHub [82].

C. Modificaciones a util-lorawan-packets

La librería `util-lorawan-packets` es relativamente fácil de usar, pero al compilarse automáticamente en Arduino IDE surgieron dos fallas. La primera falla fue un cúmulo de errores sobre funciones no referenciadas, por ejemplo, `undefined reference to `lw_join_mic``. Las funciones de los errores están definidas en una carpeta “crypto” en el archivo `lw_crypto.h`, pero, aunque estuviera escrito el `#include "crypto/lw_crypto.h"` Arduino IDE no lo tomaba como válido. Investigando parecía ser un problema de Arduino IDE porque compila código C++ y la librería está escrita en C, por lo que hay cuestiones específicas en la compilación multilenguaje que podrían afectar la compilación de Arduino IDE cuando la cadena de inclusión de los `.h` pasa cierto nivel. Hubo dos soluciones al respecto, una fue incluir todos los archivos `.c` de la carpeta “crypto” dentro del archivo con conflicto `lw_packets.c` y análogo para los `.h`, un tanto engorroso pero funcional. La segunda y más prolija solución fue mover los archivos de “crypto” un nivel más arriba junto con `lw_packets.c` y cambiando la ruta del `#include` a solamente `lw_crypto.h`.

La otra falla se dio con un par de archivos incluidos con rutas no existentes en la librería:

```
#include <github.com/lobaro/c-utils/datetime.h> en lw_packets.h y
#include <github.com/lobaro/c-utils/lobaroAssert.h> en lw_crypto.c.
```

Para el primer archivo solo fue necesario comentar/eliminar el `#include` y agregar la siguiente definición del tipo que se incluía en el archivo que no se tiene: `typedef int64_t Time_t`. Para el segundo archivo se precisó comentar/eliminar el `#include` y en su lugar agregar estas dos líneas para redefinir el `assert`:

```
#include <assert.h>
#define lobaroASSERT(x) assert(x)
```

Las últimas modificaciones no son de fallas sino de habilitar funcionalidad. Para gestionar mensajes de *uplink* en la librería en la función `LoRaWAN_UnmarshalPacket()` hay que definir una constante `LW_SUPPORT_NETWORK_SERVER` que se interpreta como el soporte en modo NS. Para esto solo debe cambiarse el valor de la constante de 0 a 1 en su respectivo `#define` de `lw_packets.c`. Sin embargo, hay un error en el código que asume que en el proceso de *unmarshal* la constante `LW_SUPPORT_NETWORK_SERVER` está en 0. Esta asunción habilita tener una variable `useAFCntDwn` definida dentro de un bloque que permite utilizarla más adelante, pero que si `LW_SUPPORT_NETWORK_SERVER` está en 1, el bloque se cierra modificando el alcance de la variable y anulando su uso posteriormente. La solución fue **mover** la declaración de la variable a una posición anterior en el código sin que la lógica se vea afectada. Así se verían las líneas cercanas de código luego del cambio:

```
1     ...
2     uint32_t currFCnt32;
3     bool uplink = false;
4
5     // CHANGED: Moved this out of the if to make it work
6     // when LW_SUPPORT_NETWORK_SERVER is defined
7     bool useAFCntDwn = false;
8     #if LW_SUPPORT_NETWORK_SERVER
```

```
9
10  if (packet->MHDR.type == MTYPE_CONFIRMED_DATA_UP ||
11      packet->MHDR.type == MTYPE_UNCONFIRMED_DATA_UP) {
12      ...
```

D. Estructura de carpetas de la entrega

```
/
├── Contramedidas para la manipulación maliciosa de dispositivos en LoRaWAN.pdf
├── Código
│   ├── fixed-payload-sender
│   │   └── fixed-payload-sender.ino
│   ├── jammer
│   │   └── jammer.ino
│   ├── mitm-poc
│   │   └── mitm-poc.ino
│   ├── raw-sender
│   │   ├── raw-sender.ino
│   │   └── raw-serial.py
│   └── README.md
├── Modelos SHARPE
│   ├── LoRaWAN.rgl
│   ├── LoRaWAN_AbusoDeComandosMAC.flt
│   ├── LoRaWAN_AccesoAdministrativoAlDispositivo.flt
│   ├── LoRaWAN_AccesoRemotoAdministrativo.flt
│   ├── LoRaWAN_AlterarComportamientoDeDispositivo.flt
│   ├── LoRaWAN_CargaDeFirmwareCorrupto.flt
│   ├── LoRaWAN_DejarFueraDeComunicacionAlDispositivo.flt
│   ├── LoRaWAN_EvitarEnvioRepcionDeMensaje.flt
│   ├── LoRaWAN_ForzarReenvioDeJR.flt
│   ├── LoRaWAN_ModificacionIndebidaDeDatos.flt
│   ├── LoRaWAN_ModificacionIndebidaDeMensajesSalientes.flt
│   ├── LoRaWAN_ObtenerClavesDeSesion.flt
│   ├── LoRaWAN_RobarClavesDeDispositivo.flt
│   ├── LoRaWAN_SuplantacionDeDispositivo.flt
│   └── README.md
```

En la raíz se encuentra el PDF de este informe. La carpeta `Código` contiene los `.ino` que fueron cargados en los SparkX, también se incluye un archivo `README.md` que describe brevemente cada `.ino` y da algunas explicaciones específicas a tener en cuenta al compilar o cambiar el comportamiento en el código.

La carpeta `Modelos SHARPE` contiene el archivo del proyecto SHARPE y los árboles de falla para todos los árboles de ataque explicados en este trabajo. También incluye un `README.md` que explica esto mismo y un detalle sobre la asignación de probabilidades de falla.