Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Análisis y *Pentesting* de la Tecnología Bluetooth

Analysis and Pentesting of Bluetooth Technology

Enrique Manuel Pedroza Castillo

La Laguna, XXX de 2021

Dña. **Pino Caballero Gil**, con **N.I.F** 45.534.310-Z, Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

Dña. **Alba Cruz Torres**, con **N.I.F** 79.073.834-H, Contratada por Proyecto de Investigación en la Universidad de la Laguna, como cotutora.

CERTIFICAN

Que la presente memoria titulada:

"**Análisis y *Pentesting* de la Tecnología Bluetooth**"

ha sido realizada bajo su dirección por **D. Enrique Manuel Pedroza Castillo** con **N.I.F 79.095.642-D**

Agradecimientos

A la profesora Pino Caballero Gil, por darme la oportunidad de realizar el presente trabajo, así como el asesoramiento y guía para poder autosuperarme con este gran reto.

Especialmente **a Miguel Ángel y Carolina** quienes han estado en las horas más bajas conmigo y me han mostrado su apoyo incondicional siempre que les he necesitado, dándome las fuerzas suficientes para seguir mejorando con cada reto.

Licencia

Resumen

La seguridad informática es un área esencial hoy en día, pues nuestra sociedad se encuentra en una era totalmente dependiente de la tecnología.

Debido a esta circunstancia se requiere de mecanismos robustos que permitan proteger la seguridad de la información y datos que creamos, enviamos y recibimos diariamente. Entre estos mecanismos destacan los de *pentesting*.

El *pentesting* se basa en identificar las debilidades de los sistemas informáticos para posteriormente acceder a ellos, con el objetivo de determinar cuán comprometidos están los sistemas y los datos almacenados. Gracias al *pentesting* se ponen a prueba los mecanismos de seguridad que protegen tanto los sistemas como los datos.

Una de las tecnologías de comunicación inalámbrica más usadas a día de hoy es Bluetooth (IEEE 802.15.1) y por ello es necesario aplicar mecanismos de seguridad para proteger los datos que son transferidos mediante esta tecnología.

En el presente trabajo se busca comprender cómo funcionan los protocolos de seguridad en Bluetooth, detectar posibles vulnerabilidades en su funcionamiento y realizar las implementaciones necesarias para exponer dichas vulnerabilidades con objeto de determinar cuán protegida está nuestra información.

El repositorio donde se desarrolló el proyecto se encuentra en el siguiente enlace a GitHub:
https://github.com/kenshinsamue/TFG

Abstract

Computer security is an essential area today, as our society is in a new technological age.

Due to this circumstance, robust mechanisms are required to protect the security of the information and data that we create, send and receive on a daily basis. Among them, those of pentesting stand out.

Pentesting is based on identifying computer systems weaknesses with the goal of gaining access and determining how much compromised the systems and stored data are. All this with the purpose of testing security mechanisms and keeping systems and data secure.

One of the most used wireless communication technologies today is Bluetooth (IEEE 802.15.1), therefore, it is necessary to apply security mechanisms to protect the data that is transferred through this technology.

The goal of this project is to understand how security protocols work in Bluetooth, detect possible vulnerabilities in their operation, and carry out the necessary implementations to expose these vulnerabilities in order to determine how protected our information is.

The repository where this project have been developed is in the following GitHub link:
https://github.com/kenshinsamue/TFG

Índice General

1. Introducción

1.1 Motivación

El *pentesting* es un área de la seguridad informática en la que se aplica el estudio de un determinado protocolo, sea *software* o *hardware*, para poder **entender** cómo funciona. Su propósito es **reforzar** las fortalezas, eliminar las debilidades y prevenir posibles ataques de dicho protocolo por parte de personas maliciosas.

Gracias a esta disciplina de la seguridad informática, los sistemas informáticos se hacen cada día más **robustos**, lo que contribuye a proteger nuestros datos.

En nuestra sociedad la tecnología está cada vez más distribuida, claros ejemplos de esto son los móviles, portátiles, tablets, etc., que, por lo general, usan tecnologías de comunicación inalámbrica tales como wifi o Bluetooth.

En el caso de **Bluetooth** se presenta una característica interesante ya que es una tecnología cuya seguridad en gran medida depende de la elección del método de emparejamiento, con lo cual existe una gran dependencia o confianza en que dicho mecanismo interno proteja tanto la información como la integridad de los diferentes sistemas en los que está implementado dicho protocolo.

El gran problema de este hecho es que, si de alguna forma se puede llegar a comprometer la comunicación, las tecnologías que dependen del protocolo se volverán inseguras, ya que se podrá acceder a los datos transmitidos, llegando incluso a poder acceder al propio dispositivo.

1.2 Objetivos

El objetivo del trabajo es estudiar cómo funciona la encriptación de los datos dentro de los paquetes Bluetooth y, de esta forma, ver cómo proteger nuestra información frente a personas no deseadas.

A día de hoy uno de los paradigmas más importantes y en evolución son las redes neuronales, algoritmos capaces de predecir resultados mediante la identificación de patrones.

Ambos aspectos han sido indispensables para llevar a cabo el proyecto, por un lado, el E0 para comprender cómo funciona la comunicación y transmisión de datos a través de Bluetooth y, por otro lado, cómo funcionan las redes neuronales para encontrar y hallar dichos patrones.

El objetivo es probar si el algoritmo de cifrado de Bluetooth clásico, el E0, es vulnerable a este tipo de algoritmos y si se es capaz de obtener la clave de cifrado con algún modelo de red neuronal. Para esto se ha creado un simulador del algoritmo E0 para generar varios conjuntos de claves o *datasets* que han sido usados para predecir la clave.

1.3 Estructura de la memoria

El presente trabajo está dividido en 7 capítulos:

1. Se explica la motivación y los objetivos establecidos en este trabajo.

2. Se introducen los antecedentes y el estado actual del protocolo Bluetooth y se describe el problema a tratar, así como las tecnologías y herramientas usadas para solucionar dicho problema.

3. Se explica el funcionamiento de la herramienta diseñada, incluyendo los diferentes componentes.

4. Se presentan los resultados de la herramienta.

5. Se establecen las conclusiones del proyecto.

6. Se incluye el presupuesto necesario para la creación y finalización del proyecto.

7. Se adjunta la bibliografía utilizada para la realización del proyecto.

2. Antecedentes

2.1 Estado del arte

Actualmente en el mercado casi todos los dispositivos electrónicos poseen algún protocolo de comunicación inalámbrica o, en todo caso, tienen una alternativa que sí posee dichos mecanismos de comunicación. Entre estos podemos encontrar desde los teléfonos móviles, *tablets*, *ebooks* y portátiles hasta electrodomésticos como neveras, lavadoras y aspiradoras.

Todos estos dispositivos usan las tecnologías inalámbricas para beneficiarse de su mejor característica: la comodidad. Sin embargo, al momento de enviar o recibir información lo hacen a través de ondas electromagnéticas, accesibles por cualquiera que esté dentro del rango de transmisión.

Estas ondas pueden ser captadas y decodificadas con el receptor adecuado, de modo que son altamente vulnerables a lo que comúnmente se llama ***Man In The Middle*** [1], permitiendo a un tercero acceder a la información que en un principio no se desea que pueda tener acceso.

De estos protocolos de comunicación inalámbrica, los más reconocidos y usados son wifi y Bluetooth. Este trabajo se enfoca sobre la segunda de estas tecnologías.

Bluetooth BR/EDR

Es la implementación clásica de Bluetooth. La comunicación inalámbrica de este protocolo se agrupa a través de lo que se denomina ***piconet***, esto es, una red que permite la comunicación desde dos hasta siete dispositivos [2].

Hay dos roles en la comunicación, un **maestro** que se encarga de gestionar y administrar la red y los **esclavos** que son los que siguen las directivas determinadas por el maestro.

El protocolo envía la información mediante un esquema multiplexado por el tiempo, es decir, ***Time Division Duplex*** (TDD) [3]. Esto es que la transmisión de datos es bidireccional y, para separar la señales de los mensajes enviados y recibidos, se le concede una cantidad de tiempo para que cada una de las partes de la comunicación transmitan los datos que tienen que enviar, de esta forma se evita colisiones en la emisión de datos.

Paquetes

Existen dos modos de transmisión de los paquetes: el primero es ***Basic Rate*** (**BR**) que sirve para enviar mensajes de comandos con información esencial, mientras que el segundo, ***Enhanced Data Rate*** (**EDR**), permite enviar información más compleja. Tal como se observa en la *Imagen 1*, el Paquete ***Basic Rate*** consta de tres campos que son el ***Access code***, el ***Header*** y el ***Payload***.

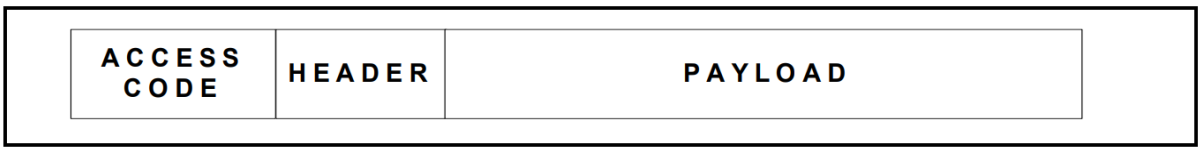


Imagen 1. Descripción de paquete Bluetooth BR

Como se puede observar en la *Imagen 2*, el ***Enhanced Data Rate*** está compuesto por el *Access Code*, el *Header*, un campo *Guard*, *Sync*, *Enhanced Data Rate Payload* y, por último, la cola.

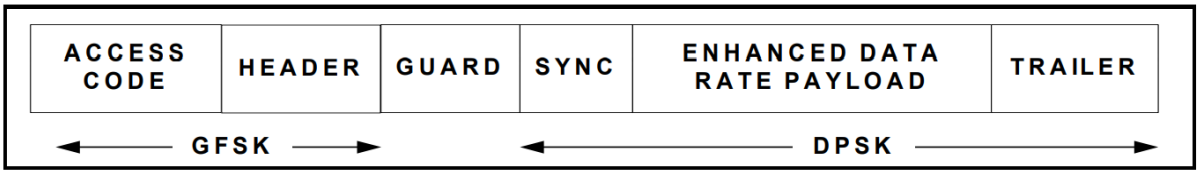


Imagen 2. Descripción paquete Bluetooth EDR

Cada uno de estos bloques consiste en lo siguiente:

Access Code

Es un código usado para la sincronización y compensación de desconexión. Se encarga además de identificar cada paquete compartido por el canal físico, de modo que todos los paquetes transmitidos por el mismo canal físico tendrán el mismo valor en este campo.

Tal como se observa en la *Imagen 3*, está formado por un preámbulo, una palabra de sincronización y una cola.

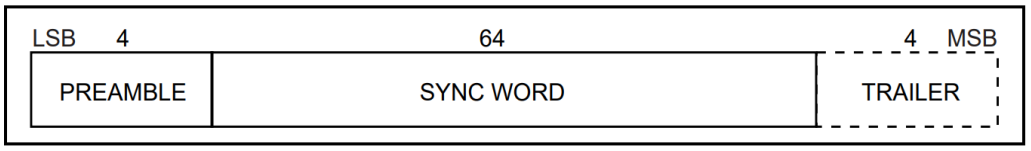


Imagen 3. Descripción del Access code

El preámbulo contiene diferentes códigos para saber el estado del dispositivo, además de compensar posibles desconexiones durante la comunicación.

Por otro lado, tenemos la palabra de sincronización que, como su nombre indica, es una cadena que permite la sincronización entre los dispositivos dentro de la *piconet*.

Header

Como se puede ver en la *Imagen 4*, la cabecera del paquete contiene información relativa al control del enlace lógico entre los dispositivos, y está dividido en 6 campos:

1. ***LT_ADDR***: Contiene la dirección del transporte lógico del paquete dentro de la *piconet.*
2. ***TYPE***: especifica el tipo del paquete.
3. ***FLOW***: especifica el flujo del paquete. El valor depende de lo que haga el receptor, de modo que si quiere parar la transmisión tiene un valor 0, mientras que si es aceptado tendrá un valor de retorno 1.
4. ***ARQN***: Es un bit que permite indicar que un paquete ha sido recibido con éxito.
5. ***SEQN***: su función es para la ordenación de los datos transmitidos.

6. ***HEC***: Este campo reservado se usa para la comprobación de errores de la cabecera (Header Error Check).

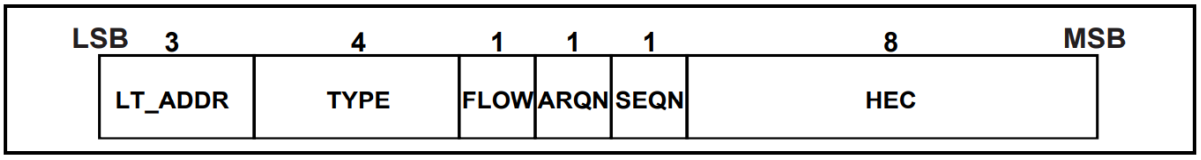


Imagen 4. Descripción de la cabecera de los paquetes

Resto de campos

El resto de los campos que hay presentes en el protocolo son: ***Guard***, que es un intervalo de tiempo de protección entre la cabecera y los datos de sincronización y es el encargado del control y sincronización de los dispositivos y *Payload* y *Enhanced Data Rate Payload*, donde ambos son campos equivalentes y sirven para la transmisión de datos ya sea sobre BR o bien EDR, respectivamente.

2.2 Introducción al problema

El protocolo *Bluetooth* para poder comunicarse con otro dispositivo utiliza el algoritmo **E0** en el cifrado de los mensajes. La clave que se usa para poder cifrar la información de la comunicación se genera de forma pseudoaleatoria en base a unos valores determinados.

Para poder generar la clave se necesita, a su vez, una clave generada por el algoritmo **E3**, la dirección **MAC** del adaptador *Bluetooth* del dispositivo maestro y el **reloj** interno de la comunicación del dispositivo maestro. De esta información solo podemos acceder de forma **pública** a los dos últimos.

En el presente proyecto se pretende valorar la posibilidad de usar un algoritmo de ***Deep Learning (DL)*** para encontrar un patrón que nos permita intentar predecir la clave usada con la información que disponemos.

2.3 Tecnologías y herramientas usadas

Bluez

Es una librería que nos da acceso a las propiedades de los dispositivos Bluetooth dentro del sistema. El programa principal, al estar escrito en ***Python,*** hace uso de la versión adaptada para este lenguaje.

Ubertooth One

Es un dispositivo hardware que permite lanzar ataques de *sniffer* contra los paquetes del protocolo Bluetooth (ver *Imagen 5*). Principalmente está hecho para la versión ***Bluetooth LE***, pero permite la recepción de paquetes de ***Bluetooth clásico (BR/EDR)***.

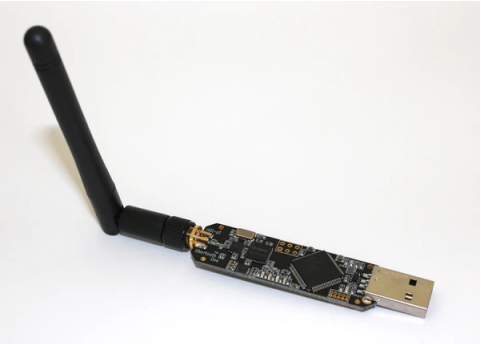


Imagen 5. Dispositivo Ubertooth one

Wireshark

Es un software analizador de paquetes que permite ver los paquetes que son recibidos, así como la información que contiene cada uno de los campos. Es capaz de leer múltiples protocolos como *TCP*, *UDP*, *HTTP*, entre otros. Esta herramienta es usada para poder ver el tráfico de paquetes capturados por el ***Ubertooth*.**

Pyshar

Es una librería para el lenguaje de programación *Python* que facilita los servicios y datos de Wireshark para dicho lenguaje.

Android Studio

Es un ***IDE*** (Entorno de Desarrollo Interactivo) para aplicaciones móviles, como su nombre dice en particular para la plataforma ***Android***. La aplicación creada ha sido implementada usando esta herramienta con el lenguaje de programación ***Java***.

Pytorch

Es un *framework*, principalmente para *Python*, que permite tener acceso a diferentes recursos usados en el campo del *DL*, tales como neuronas, capas, funciones de activación. entre otros.

3. Análisis realizado

Debido a la envergadura del proyecto, se decidió dividirlo en 5 partes:

1. **Cifrado del E0**: se analiza el funcionamiento del cifrado del protocolo Bluetooth, así como la generación de claves de cifrado.

2. **Deep Learning**: se investiga en qué consiste el algoritmo de **DL** y las diferentes funcionalidades implementadas.

3. **Bluetooth Chat**: se explica qué procesos de comunicación realiza esta aplicación para poder permitir tanto la conexión como el envío de mensajes entre ambos dispositivos conectados.

4. **Sniffing**: en la cual se detalla cómo se puede, a través del dispositivo **Ubertooth One,** obtener los paquetes de Bluetooth clásico y lo que ellos contienen, así como el proceso de configuración.

5. **Pentesting:** mediante la unificación de los apartados anteriores. Se detalla el papel que tiene cada parte del proyecto de forma individual y cómo se ha usado para obtener el resultado final del proyecto.

3.1 Cifrado E0

Bluetooth [4] utiliza el algoritmo E0 [5] para generar una secuencia cifrante, que es una cadena binaria que se suma al mensaje que se va a transmitir. Tanto el dispositivo emisor como el receptor usan la información del dispositivo **maestro** para generar la clave cifrante. En ambos casos, la operación de la suma corresponde a la operación XOR (ver *Imágenes 6 y 7*).

Input A	Input B	XOR Output
0	0	0
0	1	1
1	0	1
1	1	0

Imagen 6. Tabla operación XOR

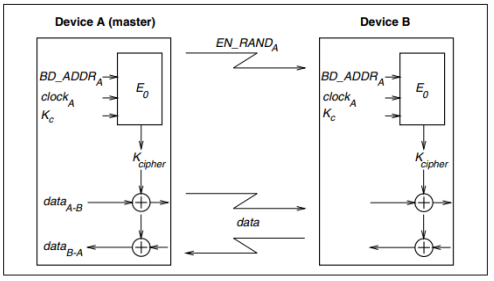


Imagen 7. Descripción del cifrado de E0

Creación de la clave cifrante con E0

Para este apartado daremos por hecho que tenemos el valor de la clave secreta compartida *kc.* La primera tarea que se debe realizar es la creación de los vectores de inicialización y esto se hace a partir de la información que disponemos. Teniendo esto en cuenta, se aplica el esquema mostrado en la *Imagen 8*:

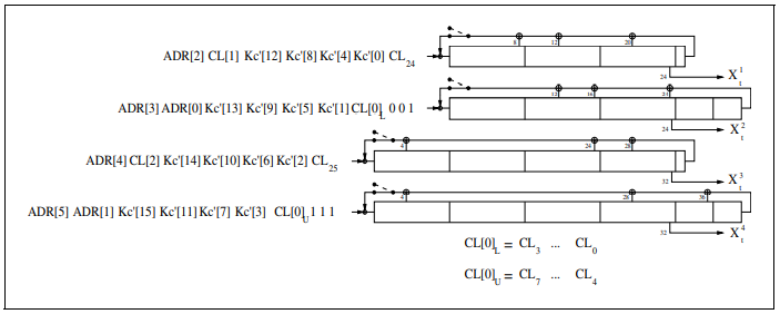


Imagen 8. Composición de los vectores de inicialización de los LFSR

Como podemos ver, los vectores se generan en base a determinados *bytes* pertenecientes a los parámetros que vienen descritos en la *Imagen 9*.

Para saber cuál de los valores corresponde a los descritos en la imagen, es indispensable saber que se usa como bit más significativo el de la izquierda, con lo que, si usamos el ejemplo de la *Imagen 9*, la dirección original la leeremos de la siguiente forma: “1B:0F:56:94:7F:2C”.

Otra consideración en cuanto a la nomenclatura está dentro del apartado de los relojes, ya que podremos ver en la notación tanto CL_L como CL_U , entre otros :

- CL_L es el primer hexadecimal (4 primeros bits) del reloj (CL[0]).
- CL_U es el segundo hexadecimal (CL[1]).
- CL_{24} y CL_{25} son los bits 24 y 25 respectivamente del reloj, del segundo hexadecimal de CL[4].

```
Initial values for the key, pan address and clock

K'c4[0]   = 21   K'c4[1]   = 87   K'c4[2]   = F0   K'c4[3]  = 4A
K'c4[4]   = BA   K'c4[5]   = 90   K'c4[6]   = 31   K'c4[7]  = D0
K'c4[8]   = 78   K'c4[9]   = 0D   K'c4[10]  = 4C   K'c4[11] = 53
K'c4[12]  = E0   K'c4[13]  = 15   K'c4[14]  = 3A   K'c4[15] = 63

Addr4[0] = 2C   Addr4[1] = 7F   Addr4[2] = 94
Addr4[3] = 56   Addr4[4] = 0F   Addr4[5] = 1B

CL4[0]  = 5F   CL4[1]  = 1A   CL4[2]  = 00   CL4[3]  = 02
```

Imagen 9. Valores iniciales para la creación de la clave

Siguiendo el ejemplo de la *Imagen 9*, habrá que crear los 4 vectores inicializadores que usaremos posteriormente para llenar los LFSR. El resultado lo vemos en la *Imagen 10*:

```
Primer vector: 0x12835c0f17442
Segundo Vector: 0x2b160a86c843f9
Tercer Vector: 0x1e00749863e1
Cuarto Vector: 0xdbfb1a9e8252f
```

Imagen 10. vectores de inicialización de los LFSR

Con los vectores se inicializan los cuatro LFSR, insertando primero los bits menos significativos. Cuando se llegue a los ciclos 26, 32, 34 y 39 se aplicará la **retroalimentación** (como viene especificado en la *Imagen 8*), para el vector correspondiente.

De esta forma, cuando lleguemos al ciclo 26 aplicaremos la retroalimentación al primer LFSR, cuando lleguemos al 32 se aplicará además al segundo y así sucesivamente. De esta forma, mientras un determinado LFSR se retroalimenta, el resto simplemente es cargado desde el vector (ver *Imagen 11*).

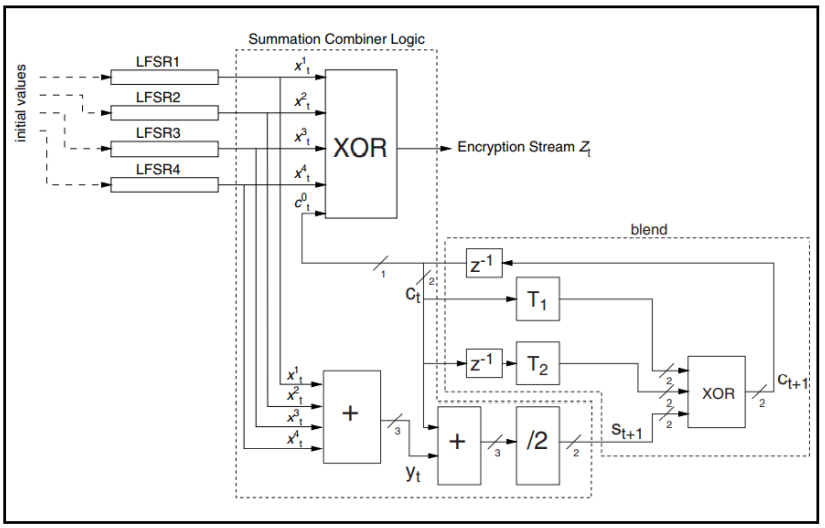


Imagen 11. Operaciones para crear la clave

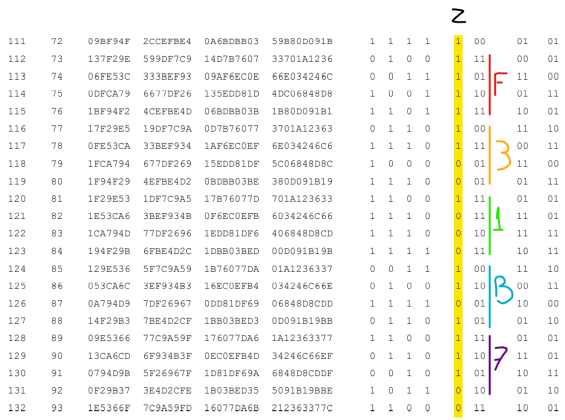
Cuando todos los LFSR estén cargados, seguiremos las operaciones descritas en la *Imagen 11* durante otros 200 ciclos más. Mientras se ejecutan tendremos en cuenta lo siguiente:

- El valor de las ***x*** de los LFSR durante los diversos ciclos se corresponde al valor del bit en la posición 24 (para los primeros dos LFSR) y 32 (para los últimos dos).
- El registro z^{-1} se refiere al valor de salida del *blend* del ciclo anterior.
- El valor de salida de z^{-1} se usa para operar con los registros T_1 y T_2. El primero usa la salida del ciclo anterior, sin embargo, T_2 usa el valor de la salida de hace **dos ciclos** (ver *Imagen 12*).

x	$T_1[x]$	$T_2[x]$
10	10	01
11	11	10

Imagen 12. Operaciones de los Registros T_1 y T_2

Después de realizar los 200 ciclos, se obtendrá el valor de la clave temporal de los últimos 128 bits, con lo que, a partir de la iteración 71 se guardarán los resultados. Podremos obtener la clave siguiendo el formato mostrado en las *Imágenes 13 y 14*:



						z				
111	72	09BF94F	2CCEFBE4	0A6BDBB03	59B80D091B	1 1 1 1	1	00	01	01
112	73	137F29E	599DF7C9	14D7B7607	33701A1236	0 1 0 0	1	11	00	01
113	74	06FE53C	333BEF93	09AF6EC0E	66E034246C	0 0 1 1	1	01	11	00
114	75	0DFCA79	6677DF26	135EDD81D	4DC06848D8	1 0 0 1	1	10	01	11
115	76	1BF94F2	4CEFBE4D	06BDBB03B	1B80D091B1	1 1 0 1	1	11	10	01
116	77	17F29E5	19DF7C9A	0D7B76077	3701A12363	0 1 1 0	1	00	11	10
117	78	0FE53CA	33BEF934	1AF6EC0EF	6E034246C6	1 1 1 0	1	11	00	11
118	79	1FCA794	677DF269	15EDD81DF	5C06848D8C	1 0 0 0	0	01	11	00
119	80	1F94F29	4EFBE4D2	0BDBB03BE	380D091B19	1 1 1 0	0	01	01	11
120	81	1F29E53	1DF7C9A5	17B76077D	701A123633	1 1 0 0	1	11	01	01
121	82	1E53CA6	3BEF934B	0F6EC0EFB	6034246C66	1 1 1 0	0	11	11	01
122	83	1CA794D	77DF2696	1EDD81DF6	406848D8CD	1 1 1 0	0	10	11	11
123	84	194F29B	6FBE4D2C	1DBB03BED	0DD091B19B	1 1 1 1	0	11	10	11
124	85	129E536	5F7C9A59	1B76077DA	01A1236337	0 0 1 1	1	00	11	10
125	86	053CA6C	3EF934B3	16EC0EFB4	034246C66E	0 1 0 0	1	10	00	11
126	87	0A794D9	7DF26967	0DD81DF69	06848D8CDD	1 1 1 1	0	01	10	00
127	88	14F29B3	7BE4D2CF	1BB03BED3	0D091B19BB	0 1 1 0	1	01	01	10
128	89	09E5366	77C9A59F	176077DA6	1A12363377	1 1 0 0	1	11	01	01
129	90	13CA6CD	6F934B3F	0EC0EFB4D	34246C66EF	0 1 1 0	1	10	11	01
130	91	0794D9B	5F26967F	1D81DF69A	6848D8CDDF	0 0 1 0	1	01	10	11
131	92	0F29B37	3E4D2CFE	1B03BED35	5091B19BBE	1 0 1 1	0	10	01	10
132	93	1E5366F	7C9A59FD	16077DA6B	212363377C	1 1 0 0	0	11	10	01

F 3 1 B 7

Imagen 13. Registro de los resultados y estado de los diferentes registros y LFSR

```
238   199    1D9BCCF   58FF53AB   1EC3A7354
239   200    1B3799E   31FEA756   1D874E6A8

Z[0]  = 3F
Z[1]  = B1
Z[2]  = 67
Z[3]  = D2
Z[4]  = 2F
Z[5]  = A6
Z[6]  = 1F
Z[7]  = B9
Z[8]  = E6
Z[9]  = 84
Z[10] = 43
Z[11] = 07
Z[12] = D8
Z[13] = 1E
Z[14] = E7
Z[15] = C3
```

Imagen 14. Resultado de los ciclos de ejecución

Una vez tengamos el valor de la clave temporal, se cargarán los diferentes hexadecimales en los correspondientes LFSR tal y como se indica en la *Imagen 15*, con lo que se obtendrá como resultado los LFSR cargados de la forma mostrada en la *Imagen 16*:

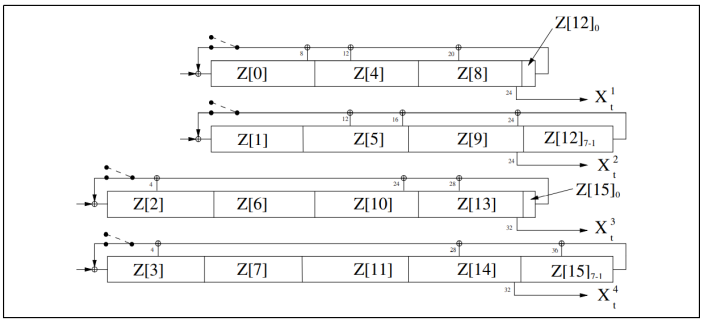


Imagen 15. Reinicialización de los LFSR en función de la clave temporal

```
LFSR1  <= 0E62F3F
LFSR2  <= 6C84A6B1
LFSR3  <= 11E431F67
LFSR4  <= 61E707B9D2
C[t+1] <= 00
```

Imagen 16. Reinicialización de los LFSR

Por último, haremos las últimas iteraciones del protocolo en función de cuántos bits queremos tener en nuestra clave, con lo cual ejecutaremos 128 ciclos más, con el resultado mostrado en la *Imagen 17*.

```
PS E:\TFG\BT> python3.exe .\E0.py
0x2999f607fde02ea4cc9c1b8503a59429
```

Imagen 17. Clave resultante de 128 bits

3.2 Deep Learning

3.2.1 Conceptos básicos

El DL forma parte del aprendizaje automático y, de hecho, puede considerarse una evolución del *Machine Learning (ML),* que pertenece al campo de la Inteligencia Artificial (IA). Dado que ambos términos pueden ser fácilmente confundidos entre sí [6], vemos qué caracteriza a cada uno.

La Inteligencia Artificial utiliza algoritmos que tienen por objetivo realizar operaciones comparables a las que realiza la mente humana, entre las que están el aprendizaje y el razonamiento lógico.

Los algoritmos de ML son un tipo de inteligencia artificial, que tienen como objetivo generar el resultado más óptimo para un problema dado y lo consigue gracias a que es capaz de modificar su propio comportamiento para aproximarse a un mejor resultado.

Entre los algoritmos de **ML** destacan los de **DL**, que se caracterizan por utilizar una red que consta de diferentes capas de nodos denominados "neuronas", que están interconectadas entre sí de capa en capa (ver *Imagen 18*). A este conjunto de capas y neuronas interconectadas se le denomina **red neuronal**.

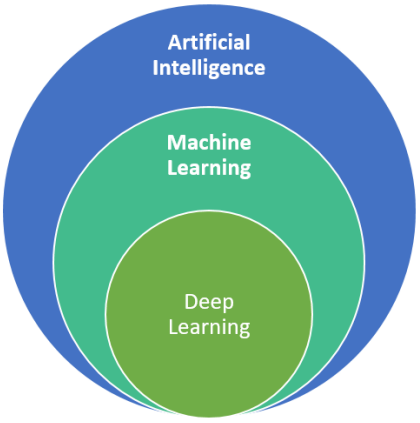


Imagen 18. Diferencias entre IA, ML y DL

3.2.2 Estructura de las redes neuronales

El DL tiene como objetivo predecir valores o situaciones no aprendidas basados en patrones de valores aprendidos previamente con algún grado de similitud. Como se ha mencionado anteriormente, este tipo de algoritmos están compuestos principalmente por capas y neuronas. Existen tres tipos de capas [7]:

1. **La capa de entrada**: es la primera capa de la red además de ser la que recibe los *inputs* de la red.

2. **La capa de salida**: es la última capa de la red y, por lo tanto, es la que nos da el resultado de los diferentes inputs de la capa de entrada.

3. **Las capas ocultas**: es un conjunto de capas intermedias que agregan una cierta complejidad y se encargan de recibir los resultados de la **capa de entrada** e irlos llevando hasta finalmente dárselo como *input* a la **capa de salida.**

Dentro de cada capa existen dos tipos de neuronas: las neuronas normales y las *BIAS* (que actúan a forma de sesgo). Las primeras tienen como input el resultado de las neuronas de la capa anterior o las que reciben los inputs de la red (en el caso de las que están en la capa de entrada), mientras que las neuronas *BIAS* son neuronas que no tienen inputs, modifican solo el peso proporcional del valor de salida.

Entre dos capas hay una serie de enlaces que permiten conectar las diferentes neuronas entre sí. Cada enlace tiene un peso que representa la importancia del valor de salida de una neurona al ser enviado a otra neurona de la siguiente capa.

El mecanismo que permite el aprendizaje de la red neuronal es *Backpropagation*, este es un método **recursivo** que evaluará los resultados de la red, desde las capas de salida hasta la capa de entrada.

Durante esta operación se irá identificando qué neurona de la red presenta el mayor error, afectando al resultado final y, una vez se identifica la neurona, se modifican los pesos de sus enlaces, con el objetivo de disminuir el error general de la red.

De este modo, la red va aprendiendo de los errores que va teniendo en función de los valores objetivos que le facilitamos. Eventualmente, con cada corrección de cada prueba se va teniendo resultados más cercanos a los resultados deseados.

La neurona

Cada neurona recibe un conjunto de valores de entrada. A estos valores se les multiplica los respectivos pesos asociados, luego suma el resultado de cada producto y el valor de la *BIAS* de la capa anterior, dando como resultado una operación de **regresión lineal** (ver *Imagen 19*).

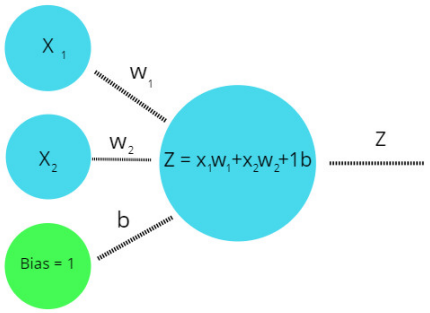


Imagen 19. Descripción de una neurona

Las capas

Son las que reúnen una colección de neuronas y les permite comunicarse con las neuronas de las capas anteriores y posteriores [8], las únicas excepciones a esta regla son: la primera, la capa de entrada caracterizada por recibir los valores de entrada directamente y la última, siendo la capa de salida caracterizada por facilitar los valores de salida (ver *Imagen 20*).

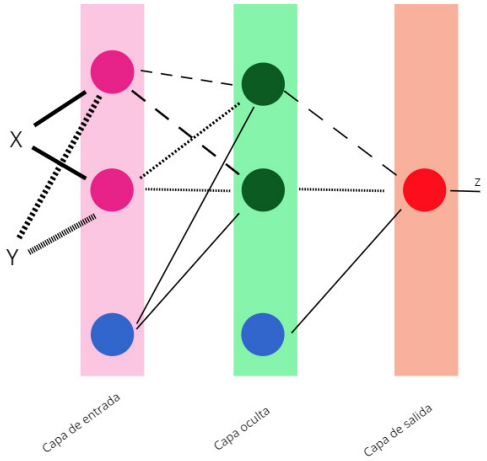


Imagen 20. Descripción de la conexión entre neuronas

La red neuronal

La red neuronal, formada por el conjunto de los elementos anteriores, gestiona el funcionamiento de cada uno de sus componentes. Las redes neuronales tienen dos formatos de funcionamiento: **Entrenamiento** y **Ejecución**.

Funciones de la red neuronal

Funciones de activación

Aunque una neurona es capaz de realizar la operación de regresión lineal en función de los valores de entrada, necesita una función de transformación que aplica un grado de distorsión al resultado de la neurona con el objetivo de evitar que la red neuronal colapse y su comportamiento sea equivalente al de una única neurona.

Existen múltiples funciones de activación, cada una con sus características y rangos de valores (ver *Imágenes 21* a *26*). Las más destacables a mencionar son:

1. La función sigmoide cuyos valores están dentro del rango [0,1].
2. La función ReLU cuyos valores están dentro del rango [0, inf).
3. La función tangente que tiene un rango de valores [-1,1].

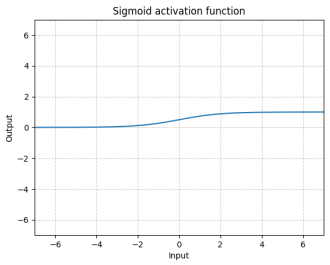


Imagen 21. Función Sigmoide

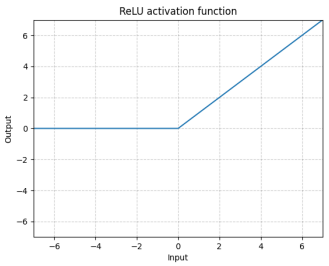


Imagen 22. Función ReLu

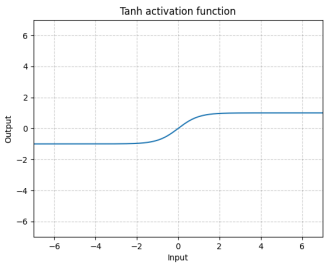


Imagen 23. Función Tangente

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + \exp(-x)}$$

Imagen 24. Fórmula de la función Sigmoide

$$\text{ReLU}(x) = (x)^+ = \max(0, x)$$

Imagen 25. Fórmula de la función ReLu

$$\text{Tanh}(x) = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

Imagen 26. Fórmula de la función Tangente

La elección de la función de activación se realiza en base a las características del problema a tratar. Esta función nos sirve, además, en el entrenamiento de la red neuronal cuando se calcula el gradiente, que nos indica hacia dónde tiene que crecer o decrecer el peso de un determinado enlace entre dos neuronas, para reducir el error general y, como consecuencia, aproximarse al resultado deseado.

Funciones de error

Las funciones de error son las que permiten a la red hacer una evaluación de la diferencia entre el valor objetivo dado y el resultado actual de la red neuronal. Una vez detectado cuáles son estas diferencias, usará un optimizador con el objetivo de minimizar el error final de la red al menor costo de rendimiento posible.

Para poder elegir la función de error que más se ajustaba al problema, se optó por hacer cuatro pruebas con diferentes modelos y evaluar los diferentes resultados.

Los modelos usados en cada una de las pruebas son los mostrados en las *Imágenes 27* a *30.*

En cada gráfica mostrada en las *Imágenes 31* a *38* nos encontramos, en el eje de las X, los epoch (número de repeticiones del entrenamiento del modelo), mientras que en el eje de las Y tenemos el valor de error. El valor debería estar entre [0,1], siendo el 100% de error representado por la unidad.

Por último, tenemos la línea azul que representa los resultados del modelo con muestras dentro del *dataset* de entrenamiento, mientras que la línea naranja representa los resultados del modelo con muestras dentro del *dataset* de evaluación. Ambos *datasets* se ven en la *Imagen 39*.

Modelo

```
model = nn.Sequential(
  nn.Linear(n_inputs,100),
  nn.ReLU(),
  nn.Dropout(p=0.2),
  nn.Linear(100,n_outputs),
  nn.ReLU()
)
```

Imagen 27: Modelo 1

```
model = nn.Sequential(
  nn.Linear(n_inputs,500),
  nn.ReLU(),
  nn.Dropout(p=0.2),
  nn.Linear(500,n_outputs),
  nn.ReLU()
)
```

Imagen 28: Modelo 2

```
model = nn.Sequential(
  nn.Linear(n_inputs,250),
  nn.ReLU(),
  nn.Dropout(p=0.2),
  nn.Linear(250,250),
  nn.ReLU(),
  nn.Dropout(p=0.2),
  nn.Linear(250,n_outputs),
  nn.ReLU()
)
```

Imagen 29. Modelo 3

```
model = nn.Sequential(
  nn.Linear(n_inputs,50),
  nn.ReLU(),
  nn.Dropout(p=0.2),
  nn.Linear(50,50),
  nn.ReLU(),
  nn.Dropout(p=0.2),
  nn.Linear(50,n_outputs),
  nn.ReLU()
)
```

Imagen 30. Modelo 4

MSE

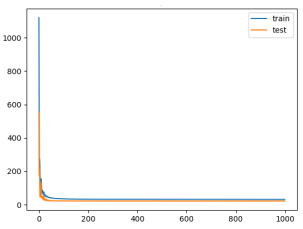


Imagen 31. Resultados MSE para la prueba 1

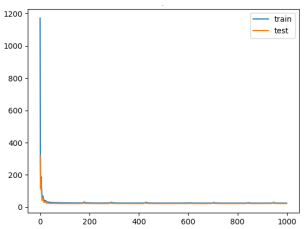


Imagen 32. Resultados MSE para la prueba 2

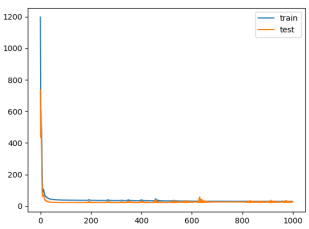


Imagen 33. Resultados MSE para la prueba 3

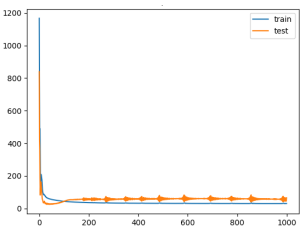


Imagen 34. Resultados MSE para la prueba 4

BCE

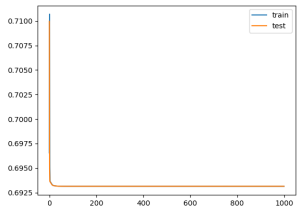


Imagen 35. Resultados BCE para la prueba 1

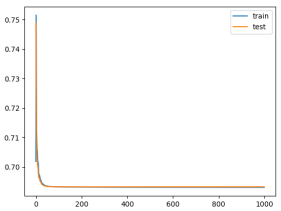


Imagen 36. Resultados BCE para la prueba 2

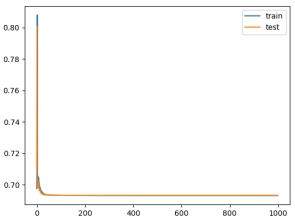


Imagen 37. Resultados BCE para la prueba 3

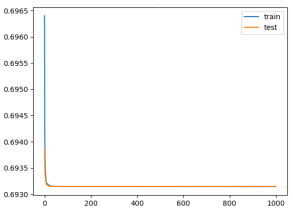


Imagen 38. Resultados BCE para la prueba 4

Como se puede apreciar, aunque todos los resultados demuestran un comportamiento similar, al fijarnos en los valores de error, se puede ver que la función de error que tiene mejores resultados es BCE (*Binary Cross-Entropy*). En consecuencia, se puede ver que aparece un problema de tipo clasificación, que se puede deber a los siguientes factores:

1. Los datos que usamos en los *datasets* son valores binarios, así como los resultados que deseamos obtener.

2. El problema que se quiere tratar busca obtener resultados que solo pueden ser de un tipo u otro exclusivamente (1 o 0), eliminando la posibilidad de tener resultados decimales o intermedios.

Optimizadores

Son funciones que permiten ayudar a las funciones de error a reducir el número de iteraciones para obtener resultados más óptimos, así como modificar los valores de las de los pesos de los enlaces de las neuronas reduciendo el tiempo de entrenamiento del modelo de la red neuronal.

Para llevar a cabo su tarea utiliza diversos parámetros, entre ellos podemos encontrar el *learning rate* que describe qué tan drásticas son las correcciones a los pesos de los enlaces en la red, los pesos de las conexiones entre neuronas u otros.

De estas funciones existen múltiples tipos y de las más importantes se puede destacar tanto Adam como SGD (*Stochastic Gradient Descent* o descenso gradiente estocástico). En este trabajo optamos por Adam como optimizador.

3.2.3 Entrenamiento

Este modo permite facilitar los valores de entrada en conjunto de sus respectivos resultados esperados. Para ello, se divide el *dataset* en dos partes: la primera, que será usada en el entrenamiento, mientras que la segunda se utilizará en las pruebas.

```
# *------------- Separacion de datos en entrenamiento y prueba ------------------*
X_train, X_test, y_train, y_test =train_test_split(x,y,test_size=0.2, train_size=0.8,random_state=1)

n_samples,n_inputs = X_train.shape
n_outputs = y_train.shape[1]
# *------------- /Separacion de datos en entrenamiento y prueba ------------------*
```

Imagen 39. División de dataset

Una vez el *dataset* ha sido dividido, se le aplican las operaciones especificadas por la estructura del modelo compuesta por las capas de neuronas, las capas de funciones de activación, las capas de descarte, entre otros.

Cuando se terminan de ejecutar las diferentes operaciones y se obtiene un resultado, este es comparado con los resultados deseados. Una vez realizada esta comparación se utilizan en conjunto la función de error y el optimizador para minimizar la diferencia entre los resultados obtenidos y los esperados, tal y como aparece en la *Imagen 40*.

```
for epoch in range(num_epoch):
  # fordward y calcula el loss
  y_predicted = model(X_train)
  loss = funcion_loss(y_predicted,y_train)
  # back propagation
  loss.backward()
  # actualizacion
  optimizer.step()
  optimizer.zero_grad()
```

Imagen 40. Proceso de entrenamiento de la red neuronal

3.2.4 Pruebas

En este modo se usan muestras que están fuera del grupo de entrenamiento, esto con el objetivo de que la representación de los datos no se vea afectada por un aprendizaje basado en memorización, sino que esta sea en base a la abstracción del problema planteado.

En la parte de pruebas lo que se hace es verificar el estado actual del modelo de la red neuronal, antes de aplicar cualquier tipo de operación relacionada con la modificación de los pesos de las conexiones. Esto con la intención de verificar el valor resultante de la función de error en el estado actual del modelo. El código correspondiente se puede ver en la *Imagen 41*.

```
model.eval()
target = model(X_test)
loss_t = funcion_loss(target,y_test)
loss_array.append(loss.item())
val_loss_array.append(loss_t.item())
model.train()
```

Imagen 41. Operaciones del modo de pruebas

3.2.5 Creación del *dataset*

Los algoritmos de **DL** necesitan de un conjunto de datos para evaluar sus resultados. A este tipo de entrenamiento se le conoce como entrenamiento supervisado. Estos datos se suelen organizar en lo que se llama ***datasets***.

Los *datasets* tienen múltiples formatos, no obstante, el elegido para el proyecto es ***CSV*** (Comma Separated Values), lo cual significa que separaremos los valores por comas. Nuestro *dataset* tendrá la información que disponemos para poder ejecutar el E0, así como la clave resultante.

De modo que los valores que guardaremos serán los siguientes:

1. La dirección MAC del dispositivo Bluetooth.
2. Una clave de 128 bits.
3. El valor del reloj.
4. La clave resultante.

Para que la red pueda ser entrenada se genera un diccionario, el cual es creado gracias al generador de claves E0, que se comentó anteriormente. El formato de este diccionario generado en primera instancia se puede observar tanto en la *Imagen 42* como en la *Imagen 43*.

MAC Bluetooth	CK	CLK	Z

Imagen 42. Formato del diccionario

Recordemos que los campos presentan el siguiente orden:

1. Dirección del adaptador.
2. Clave interna proporcionada por E3.
3. CLK es el reloj interno del adaptador maestro.
4. Clave resultante.

```
BDADDR,CK,CLK,Z
54:EC:D6:06:54:46,F4E167BDC37481668E40397122FA98F2,6F291859,35C68361E495A73E5FA7C1443987ADC1
D7:6D:C1:9E:19:E8,C55C4345A664D73EE23F81750B38CA29,E9282026,032B02DBC2A15125B9E6CE816F1A8696
04:CF:DA:67:F8:68,349CC2632AB673247C90A45C7CD7ABEE,292B0749,6238FE159CBA12BEC7BBD63119680048
46:85:C5:63:0C:63,BBF3FEF542C5C86C9463FFAFD30BA005,9EA8E32F,807317A719426B40BD7E1FA3F1AA9626
A6:71:08:9E:F0:36,495B29E71C476EDF6A40B74F2994112A,6E5B8574,144F4EF34934E9F859B2E3F6C4785B61
D5:96:D6:AE:8C:C8,CAA320ADAF8104DF2C554ED7F56EE130,2ACDEB71,2C8EE1A95D753D2EF710389AF8053BD3
```

Imagen 43. Descripción del dataset con los valores

Ante esto nos encontramos con dos circunstancias:

a. Como hemos comentado anteriormente, la clave CK no es accesible de forma pública, con lo que habrá que eliminar la columna de dicho diccionario.

b. Según la documentación [9] de *Pytorch* [10], el valor más alto que se puede usar es de 64 bits, por eso, hay que buscar una forma alternativa de representar los datos.

Para solucionar ambos inconvenientes se ha optado por eliminar, claramente, la columna de CK y usar la representación binaria de los diferentes hexadecimales que componen el resultado.

Usamos la librería *Pandas* para la manipulación de los *datasets* en formato *CSV* y, posteriormente, se aplicará el resultado del método *BitArray (ver Imágenes 44 y 45)* para convertir la cadena en un vector de bits.

```python
def BitArray(HEX):
  hexadecimal =[]
  for x in HEX:
    if x =="0":
      hexadecimal.append(0)
      hexadecimal.append(0)
      hexadecimal.append(0)
      hexadecimal.append(0)
    elif x =="1":
      hexadecimal.append(0)
      hexadecimal.append(0)
      hexadecimal.append(0)
      hexadecimal.append(1)
    elif x =="2":
      hexadecimal.append(0)
      hexadecimal.append(0)
      hexadecimal.append(1)
      hexadecimal.append(0)
```

```python
    elif x =="D" or x=="d":
      hexadecimal.append(1)
      hexadecimal.append(1)
      hexadecimal.append(0)
      hexadecimal.append(1)
    elif x =="E" or x =="e":
      hexadecimal.append(1)
      hexadecimal.append(1)
      hexadecimal.append(1)
      hexadecimal.append(0)
    elif x =="F" or x=="f":
      hexadecimal.append(1)
      hexadecimal.append(1)
      hexadecimal.append(1)
      hexadecimal.append(1)
  return hexadecimal
```

Imagen 44. Función BitArray

```python
def modificarcsv():
  cabecera = []
  cabecera.append("MAC")
  cabecera.append("CLK")
  for x in range (0,2):
    cabecera.append("Z{}".format(x))

  for i in range(1,FICHEROS+1):
    with open("{}/original/muestra{}.csv".format(PATH,i),'r') as source:
      reader = csv.reader(source)

      with open("{}/binario/muestra{}_bin_.csv".format(PATH,i),'w') as resul
        writter = csv.writer(result)
        for r in reader:
          if r == ['BDADDR','CK','CLK', 'Z']:
            cabecera = []
            for x in range(48):
              cabecera.append("MAC{}".format(x))
            for x in range(32):
              cabecera.append("CLK{}".format(x))
            for x in range (128):
              cabecera.append("Z{}".format(x))
            writter.writerow(cabecera)
          else:
            muestra = []
            MAC_= BitArray(ModificarMAC(r[0]))
            for x in MAC_:
              muestra.append(x)
            clk = BitArray(r[1])
            for x in clk:
              muestra.append(x)
            z = BitArray(r[2])
            for x in z:
              muestra.append(x)
            writter.writerow(muestra)
        result.close()
      source.close()
```

Imagen 45. Función para modificar el dataset original

De esta forma obtenemos el resultado mostrado en la *Imagen 46*:

```
MAC0,MAC1,MAC2,MAC3,MAC4,MAC5,MAC6,MAC7,MAC8,MAC9,MAC10,MAC11,MAC12,MAC13,MAC14,MAC15,MAC16,MAC17,MAC18,MAC19,MAC20,MAC21,MAC22,MAC23,MAC24,MAC25,MAC26,MAC27,MAC28,MAC29,MAC30,MAC31,MAC32,MAC33,MAC34,MAC35,MAC36,MAC37,MAC38,MAC39
,MAC40,MAC41,MAC42,MAC43,MAC44,MAC45,MAC46,MAC47,CLK0,CLK1,CLK2,CLK3,CLK4,CLK5,CLK6,CLK7,CLK8,CLK9,CLK10,CLK11,CLK12,CLK13,CLK14,CLK15,CLK16,CLK17,CLK18,CLK19,CLK20,CLK21,CLK22,CLK23,CLK24,CLK25,CLK26,CLK27,CLK28,CLK29,CLK30,CLK3
1,Z0,Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8,Z9,Z10,Z11,Z12,Z13,Z14,Z15,Z16,Z17,Z18,Z19,Z20,Z21,Z22,Z23,Z24,Z25,Z26,Z27,Z28,Z29,Z30,Z31,Z32,Z33,Z34,Z35,Z36,Z37,Z38,Z39,Z40,Z41,Z42,Z43,Z44,Z45,Z46,Z47,Z48,Z49,Z50,Z51,Z52,Z53,Z54,Z55,Z56,Z57,Z58,Z
59,Z60,Z61,Z62,Z63,Z64,Z65,Z66,Z67,Z68,Z69,Z70,Z71,Z72,Z73,Z74,Z75,Z76,Z77,Z78,Z79,Z80,Z81,Z82,Z83,Z84,Z85,Z86,Z87,Z88,Z89,Z90,Z91,Z92,Z93,Z94,Z95,Z96,Z97,Z98,Z99,Z100,Z101,Z102,Z103,Z104,Z105,Z106,Z107,Z108,Z109,Z110,Z111,Z112,Z
113,Z114,Z115,Z116,Z117,Z118,Z119,Z120,Z121,Z122,Z123,Z124,Z125,Z126,Z127
0,1,0,1,0,1,0,0,1,1,1,0,1,1,0,0,1,1,0,1,0,1,1,0,0,0,0,0,0,1,1,0,0,1,0,1,0,1,0,0,0,1,0,0,0,1,1,0,1,1,1,1,0,1,0,0,1,1,1,0,0,0,0,1,0,1,1,0,0,1,1,1,1,0,1,1,1,1,0,1,1,1,0,0,0,0,1,1,0,1,1,1,0,1,0,0,1,0,0,0,0,0,0,1,0,1,1,0,0,1,1,0,1,0,0
,0,1,1,1,0,0,1,0,0,0,0,0,0,0,0,1,1,1,0,0,1,0,1,1,1,0,0,0,1,0,0,1,0,0,0,1,0,1,1,1,1,1,0,1,0,1,0,0,1,1,0,0,0,1,1,1,1,0,0,1,0,0,1,1,0,1,1,1,1,0,0,1,0,1,0,0,1,0,0,0,1,1,0,0,0,0,1,0,1,1,0,0,1
1,1,0,1,0,1,1,1,0,1,1,0,1,1,0,1,1,1,0,0,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,1,1,0,0,1,1,1,1,0,1,0,0,0,1,1,0,0,0,1,0,1,0,1,0,1,1,1,0,0,0,1,0,0,0,0,1,1,0,1,0,0,0,1,0,1,1,0,1,0,0,1,1,0,0,1,1,0,0,1,0,0,1,1,0,1,0,1,1,1,0,0,1,1,1,1,1,0,1,1,1
,0,0,0,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,1,0,1,1,1,0,1,0,1,0,0,0,0,1,0,1,1,0,0,1,1,1,0,0,0,1,1,0,0,1,0,1,0,0,0,1,0,1,0,0,1,1,1,1,0,1,0,0,1,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,1,0
```

Imagen 46. Dataset resultante

Como podemos ver, ya la columna ha sido eliminada y los valores han sido codificados en binario dentro del rango especificado. Con esto, los diccionarios están preparados para ser usados por el modelo de DL.

3.2.6 Modelo

Para el modelo que hemos elegido se ha optado por utilizar los siguientes inputs:

1. Los bytes que componen las diferentes partes de la dirección MAC.
2. Los bytes que componen el valor final del reloj de la comunicación del dispositivo maestro.

Siguiendo el ejemplo que se usó en el apartado del E0 destacamos los componentes en la *Imagen 47*.

```
Initial values for the key, pan address and clock

K'c4[0]   = 21   K'c4[1]   = 87   K'c4[2]   = F0   K'c4[3]  = 4A
K'c4[4]   = BA   K'c4[5]   = 90   K'c4[6]   = 31   K'c4[7]  = D0
K'c4[8]   = 78   K'c4[9]   = 0D   K'c4[10]  = 4C   K'c4[11] = 53
K'c4[12]  = E0   K'c4[13]  = 15   K'c4[14]  = 3A   K'c4[15] = 63

Addr4[0] = 2C   Addr4[1] = 7F   Addr4[2] = 94
Addr4[3] = 56   Addr4[4] = 0F   Addr4[5] = 1B

CL4[0]  = 5F   CL4[1]  = 1A   CL4[2]  = 00   CL4[3]  = 02
```

Imagen 47. Datos a utilizar

Como en nuestro *dataset* tenemos juntos los datos del reloj y de la dirección MAC con la clave resultante, habrá que separarlos en dos *datasets*. Para ello, se usarán las librerías *Pandas* y *Numpy* para gestionar el *dataset*. Finalmente, se convertirá desde el formato *Numpy* a los *tensors* de *Pytorch*. Para esta tarea se creó el código mostrado en la *Imagen 48*.

```
path = "diccionario/binario/muestra1_bin_.csv"
dataset = pd.read_csv(path,skiprows=0,dtype=np.float32)
# aux_ds = pd.read_csv("diccionarios/partes/binario/muestra2_bin_.csv",skiprows=0,dtype=np.float32)
# dataset = dataset.append(aux_ds)
# *------------ Separacion entre los datasets de inputs/outputs ------------------*
headers = []
for x in range(32):
  headers.append("Z{}".format(x))

resultados=[]
for x in headers:
  resultados.append(dataset.pop(x))
result = pd.concat(resultados,axis=1,keys=headers)
# *------------ /Separacion entre los datasets de inputs/outputs ------------------*
# *------------ Conversion a un dataset de pytorch ------------------*
x = pd.DataFrame(dataset).to_numpy(dtype=np.float32)
y = pd.DataFrame(result).to_numpy(dtype=np.float32)


del(headers)
del(resultados)
del(result)

x = torch.from_numpy(x)
y = torch.from_numpy(y)
```

Imagen 48. Operaciones de carga de los datasets

Con esto tenemos dos *datasets*: el primero llamado ***X*** (que se corresponde con los datos de entrada) e ***Y*** (que son los resultados).

A continuación, habrá que separar los diferentes *datasets* para los datos destinados al entrenamiento y los datos usados en las pruebas. Como cada *dataset* consta de 1 millón de muestras, los valores serán repartidos en una proporción de 80 % de muestras de entrenamiento y 20% de muestras de prueba, tal y como se muestra en la *Imagen 39*.

Por último, se ha de especificar el modelo, la gestión de capas, la función de error, el *learning rate*, el número de *epoch* y el optimizador. Estos parámetros se especifican en la *Imagen 49*.

```
learning_rate =0.01
funcion_loss = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(),lr=learning_rate)

num_epoch = 1000
```

Imagen 49. Elección de los parámetros usados para el entrenamiento de la red neuronal

3.3 Bluetooth Chat

Para poder garantizar que los mensajes desencriptados son los obtenidos en el resultado del resto de operaciones, se ha optado por crear una aplicación Android [13], en la versión Java, de un chat entre usuarios sobre el protocolo Bluetooth.

3.3.1 Actividades

La aplicación consta de dos actividades: la primera actividad actúa a modo de home y es donde aparecerán los diferentes mensajes enviados y recibidos durante la comunicación. Tendremos un ***EditText*** en el cual el usuario podrá introducir los mensajes que quiere enviar y un botón para poder ejecutar la acción de envío del mensaje. Podemos ver una vista de la interfaz de usuario en la *Imagen 50*.

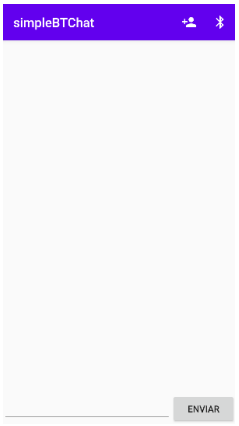


Imagen 50. Home de la aplicación

En la parte superior del menú podemos ver dos botones: el primero permite encender el Bluetooth del dispositivo, mientras que el segundo nos lleva a la segunda actividad. Estos botones aparecerán destacados en la *Imagen 51.*

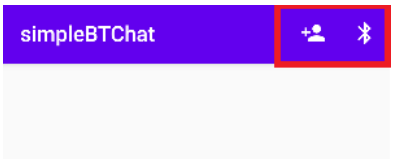
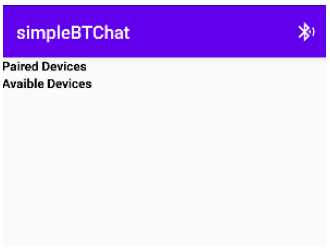


Imagen 51. Botones del home de la aplicación

La segunda actividad sirve para listar los diferentes dispositivos vinculados y los que se encuentran disponibles en el área cercana. La vista de la actividad se puede ver en la *Imagen 52,* mientras que en la *Imagen 53* se ve la misma actividad, pero con contenido de los diferentes dispositivos disponibles.



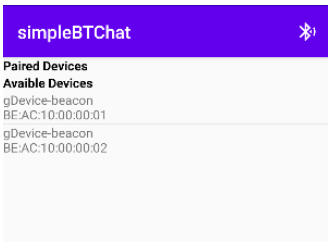


Imagen 52. Panel de dispositivos cercanos y vinculados

Imagen 53. Panel mostrando los dispositivos disponibles

Al igual que en la primera actividad, en esta segunda tenemos un menú que nos permite activar la función de scan además de hacer a nuestro dispositivo visible. Este botón aparece destacado en la *Imagen 54*.

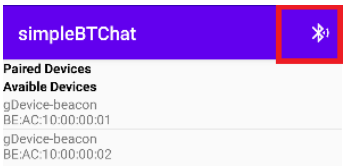


Imagen 54. Botón de búsqueda de los dispositivos Bluetooth

Al pulsar sobre un dispositivo podrá ocurrir dos eventos: el primero, si el dispositivo no ha sido emparejado, nos mostrará un mensaje para poder emparejarlo, luego irá directamente al chat. En el segundo, si ya ha sido previamente registrado y enlazado, abrirá el chat. En la *Imagen 55* se puede ver el chat en funcionamiento.

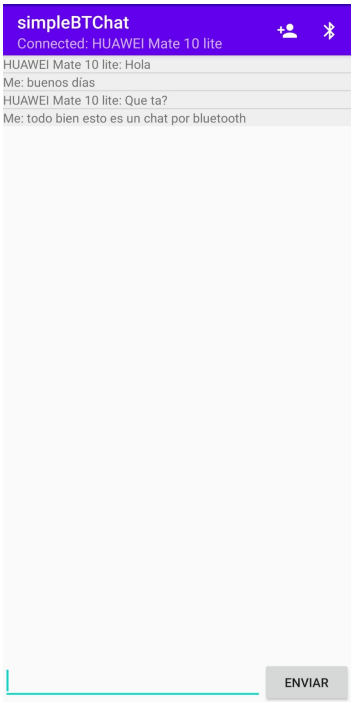


Imagen 55. Chat en funcionamiento

3.3.2 Funcionamiento

Al iniciar la aplicación se inicializan los diferentes mecanismos de Bluetooth del dispositivo, estos son tanto habilitar el Bluetooth en el sistema operativo, así como iniciar los procesos que permiten la comunicación. En nuestro programa esto se logra gracias al objeto "*ChatUtils*". Los segmentos de código más importantes de este objeto se pueden ver desde la *Imagen 56* a la *58*:

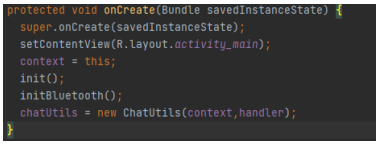


Imagen 56. Función de creación de la aplicación

Imagen 57. Inicializador de todas las partes de la aplicación

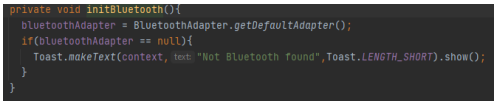


Imagen 58. Activador del Bluetooth en el dispositivo

Como vemos, la aplicación inicializa los diferentes componentes de la actividad principal que, en este caso, son:

1. El espacio reservado para el historial de mensajes.
2. El ***EditText*** donde introducimos el mensaje a enviar.
3. El botón de envío. Este además tiene un disparador de evento en el que ejecutará todo el mecanismo en caso de pulsar el botón.

En lo que se refiere a *ChatUtils*, es un objeto al que se delega la gestión de la comunicación a través de tres *Threads* de ejecución: ***AcceptThread***, ***ConnectThread*** y ***ConnectedThread***.

AcceptThread

Este hilo se ejecuta con el fin de que el dispositivo esté escuchando conexiones por parte de otro dispositivo. Filtra los mensajes recibidos por un nombre de la aplicación que usará el socket de comunicación, así como el *UUID* (Universally Unique Identifier, identificador único universal).

De esta forma, las comunicaciones que tengan esta información serán derivadas por parte del sistema operativo a nuestra aplicación. Toda la lógica del funcionamiento del AcceptThread viene especificado en las *Imágenes 59 y 60*.

Imagen 59. Constructor de AcceptThread

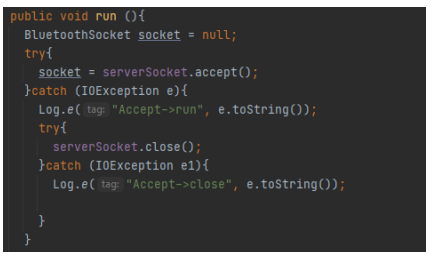


Imagen 60. Ejecutor del Thread

ConnectThread

Es un hilo de ejecución que se encarga de solicitar a un dispositivo una conexión. Este dispositivo tendrá ejecutando *AcceptThread* y le indicará que desea establecer una comunicación mediante ***RFCOMM*** que es "*Radio Frequency Communication*" y no es más

que el conjunto de protocolos de transporte que usa Bluetooth para poder comunicarse. Se puede ver cómo se especifica un socket de este tipo de comunicación en la *Imagen 61*.

Al usar *RFCOMM* estamos estableciendo que la comunicación se hará a través de Bluetooth **BR/EDR** en vez de **Low Energy**. Como podremos ver, se le facilita el valor del UUID que el terminal que está escuchando espera recibir para aceptar y abrir la comunicación.

El código que permite el manejo de errores, así como la conexión entre dispositivos está dentro de la *Imagen 62*.

```
private class ConnectThread extends Thread{
  private final BluetoothSocket socket;
  private final BluetoothDevice device;
  public ConnectThread(BluetoothDevice device){
    this.device = device;
    BluetoothSocket tmp = null;
    try{
      tmp = device.createRfcommSocketToServiceRecord(APP_UUID);
    }catch (IOException e){
      Log.e( tag: "Connect->constructor", e.toString());
    }
    socket = tmp;
  }
```

Imagen 61. Constructor de ConnectThread

```
public void run (){
  try{
    socket.connect();
  }catch (IOException e_){
    Log.e( tag: "Connect->Run", e_.toString());
    try{
      Log.d( tag: "La ejecucion del trhead ha sido interrumpida", msg: ", vamos a cerrar el socket");
      socket.close();
    }catch (IOException e){
      Log.e( tag: "Connect->CloseSocket", e.toString());
    }
    connectionFailed();
    return;
  }
  synchronized (ChatUtils.this){
    connectThread = null;
  }
  connected(socket,device);
}
```

Imagen 62. Ejecutor del Thread

En caso de que la conexión sea aceptada, se creará el último hilo (ConnectedThread), que se encarga de gestionar la conexión.

ConnectedThread

Este último hilo, como se mencionó anteriormente, es el que gestiona la conexión ya establecida entre ambos dispositivos. Al momento de crear la conexión mantendremos los sockets activos e instanciaremos los *stream*, tanto de entrada como de salida (ver *Imagen 63*).

Cuando se ejecute, estará escuchando por el próximo mensaje que entre por el canal de comunicación establecido. Esto viene especificado en la *Imagen 64*.

```
private class ConnectedThread extends  Thread{

  private final BluetoothSocket socket;
  private final InputStream inputStream;
  private final OutputStream outputStream;

  public ConnectedThread(BluetoothSocket socket){
    this.socket = socket;
    InputStream tmpin = null;
    OutputStream tmpout = null;
    try{
      tmpin = socket.getInputStream();
      tmpout = socket.getOutputStream();
    }catch (IOException e){
    }
    inputStream= tmpin;
    outputStream = tmpout;
  }
```

Imagen 63. Constructor de ConnectedThread

```
public void run(){
  byte [] buffer = new byte[1024];
  int bytes;
  while(true){
    try{
      bytes = inputStream.read(buffer);
      handler.obtainMessage(MainActivity.MESSAGE_READ,bytes, arg2: -1,buffer).sendToTarget();
    }catch (IOException e){
      connectionLost();
    }
  }
}
```

Imagen 64. Ejecutor del Thread

Además, en este hilo podremos gestionar la escritura de datos y, para ello, el método "***write***" es el que nos permite introducir un mensaje nuevo en el canal de comunicación para que sea recibido al ConnectedTrhead del dispositivo destino, el cual se ejecutará cuando se apriete el botón dentro de la actividad principal. Podemos ver el método dentro de la *Imagen 65*.

```
public void write(byte[] buffer){
  int tmp = state;
  try {
    outputStream.write(buffer);
    handler.obtainMessage(MainActivity.MESSAGE_WRITE, arg1: -1, arg2: -1,buffer).sendToTarget();
  }catch (IOException e){
    Log.e( tag: "Error al escribir",e.toString());
  }
}
```

Imagen 65. Método write de ConnectedThread

3.4 Sniffing

Para el proceso de *Sniffing* fue necesario realizar una investigación previa, puesto que los adaptadores Bluetooth, en su mayoría, no vienen preparados para estar en un modo monitor.

Un dispositivo está en modo monitor cuando se le permite rastrear las comunicaciones que están en el rango de recepción del dispositivo (sin estar conectado a la red por la que son transmitidos), por lo que para poder hacer este proceso de sniffing se requiere de un hardware especializado. En este proyecto se optó por la utilización del **Ubertooth One** [12]**.**

3.4.1 Configuración Ubertooth

Para poder configurar el Ubertooth One hay que seguir una serie de comandos, estos son:

1. Descargar y descomprimir las librerías del proyecto de Ubertooth One desde Github: https://github.com/greatscottgadgets/libbtbb/releases/tag/2020-12-R1
2. Descargar y descomprimir el repositorio del proyecto de Ubertooth One: https://github.com/greatscottgadgets/ubertooth/releases/tag/2020-12-R1
3. Instalar las herramientas y librerías del sistema operativo en base Linux (ver *Imagen 66*).

```
sudo apt install cmake libusb-1.0-0-dev make gcc g++ libbluetooth-dev wget \
pkg-config python3-numpy python3-qtpy python3-distutils python3-setuptools
```

Imagen 66. Comando de instalación de librerías y dependencias para el S.O.

4. Instalar las librerías del dispositivo (ver *Imagen 67*):

```
wget https://github.com/greatscottgadgets/libbtbb/archive/2020-12-R1.tar.gz -O libbtbb-2020-12-R1.tar.gz
tar -xf libbtbb-2020-12-R1.tar.gz
cd libbtbb-2020-12-R1
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

Imagen 67. Instalación de librerías de Ubertooth One

5. Instalar las herramientas de Ubertooth en el sistema (*ver Imagen 68*):

```
wget https://github.com/greatscottgadgets/ubertooth/releases/download/2020-12-R1/ubertooth-2020-12-R1.tar.xz
tar -xf ubertooth-2020-12-R1.tar.xz
cd ubertooth-2020-12-R1/host
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

Imagen 68. Instalación de las herramientas de Ubertooth One

6. Verificar el firmware del dispositivo con el comando ubertooth-dfu. Veremos tanto el comando como el resultado en la *Imagen 69*:

```
$ ubertooth-dfu -d bluetooth_rxtx.dfu -r
Switching to DFU mode...
Checking firmware signature
.....................................
.....................................
.............
```

Imagen 69. Verificación del firmware de Ubertooth One

7. Conceder los permisos del sistema a nuestro usuario (ver *Imagen 70)*:

Imagen 70. Modificación del grupo de usuario

Una vez ejecutado todo esto, nuestro dispositivo estará listo para ser usado, con lo cual lo siguiente que se debe realizar es la instalación y configuración de Wireshark.

3.4.2 Configuración de Wireshark

Lo primero que tendremos que hacer es agregar las dependencias a los repositorios disponibles para nuestro sistema operativo para instalar Wireshark y poder ejecutarlo (*ver Imagen 71* y *72*).

```
$ sudo add-apt-repository ppa:wireshark-dev/stable
$ sudo apt-get update
$ sudo apt-get install wireshark
```

Imagen 71. Instalación de Wireshark

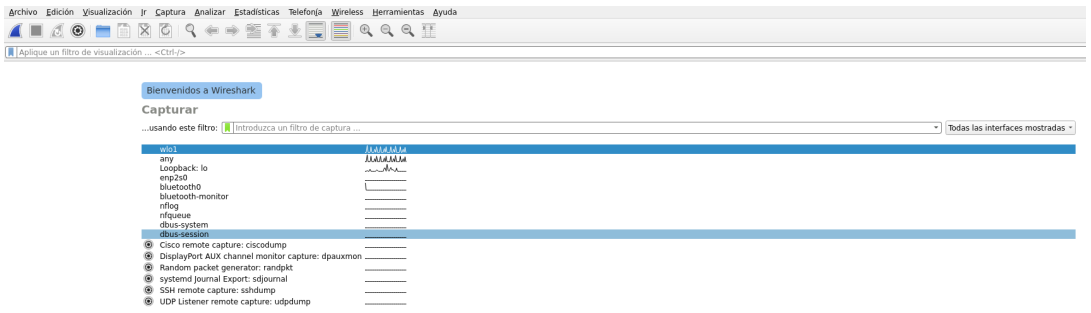


Imagen 72. Aplicación Wireshark

Cuando tengamos nuestra herramienta funcionando, lo siguiente que se hará es crear un fichero en el que se guardarán los paquetes que serán objeto de un ataque con *sniffer*. Para este ejemplo se usará un *pipe* del sistema. El comando aparece dentro de la *Imagen 73.*

```
k1k4ss0@pop-os:~$ mkfifo /tmp/pipe
k1k4ss0@pop-os:~$
```

Imagen 73. Creación del pipe del sistema

Antes de continuar con el proceso de detección de paquetes, hay que entender cómo se divide la dirección MAC de los adaptadores Bluetooth y, para eso, hay que descomponerlo en tres partes como se ve en la *Imagen 74*.

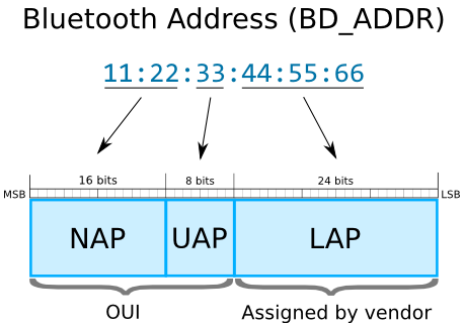


Imagen 74. Descripción de la dirección Mac

El primer segmento, **NAP** (Non-Significant Address Part), se usa para el salto en las frecuencias de sincronización durante la comunicación, así como para la **generación** de la clave con **E0**.

El segundo segmento, **UAP** (Upper Address Part), se usa para inicializar algunos algoritmos de Bluetooth. Por último, está el **LAP** (Lower Address Part), que es un valor asignado por el vendedor, de modo que identifica de forma única al adaptador.

La suma del **UAP** y del **LAP** da como resultado el **SAP** (Significant Address Part) o también la parte importante (o significativa) de la dirección MAC. Teniendo esto en cuenta, una vez tenemos nuestro fichero temporal en *tmp*, podremos ejecutar el comando de Ubertooth [11] para escuchar los paquetes que estén sobre el protocolo Bluetooth clásico. Un ejemplo de ejecución aparece en la *Imagen 75*.

```
k1k4ss0@pop-os:~$ ubertooth-rx -q /tmp/pipe -l df039f -u 49
systime=1643050955 ch=55 LAP=df039f err=1 clkn=609264 clk_offset=284 s=-71 n=-55 snr=-16
offset < CLK_TUNE_TIME
CLK100ns Trim: 4284
systime=1643050957 ch=41 LAP=df039f err=1 clkn=614836 clk_offset=2217 s=-70 n=-55 snr=-15
systime=1643050957 ch=60 LAP=df039f err=2 clkn=615888 clk_offset=2214 s=-70 n=-55 snr=-15
CLK6 = 0x28 found after 2 total packets.
```

Imagen 75. Ejecución de la herramienta ubertooth-rx

Del comando podemos ver claramente que se le especifican las siguientes instrucciones:

1. -l se refiere a LAP de la dirección MAC del adaptador Bluetooth.
2. -u se refiere al UAP de la dirección MAC del adaptador Bluetooth.
3. -q es el fichero al cual se va a escribir la información de los paquetes registrados.

3.4.3 Paquetes

Dentro de Wireshark automáticamente veremos los paquetes de Bluetooth del dispositivo que hemos especificado. De entre ellos podremos ver cómo efectivamente hay algunos que contienen datos encriptados y estos serán en los que estamos interesados en obtener para desencriptarlos. El proceso de creación de interfaz y la visualización del paquete con datos son visibles dentro de las *Imágenes 76-79*.

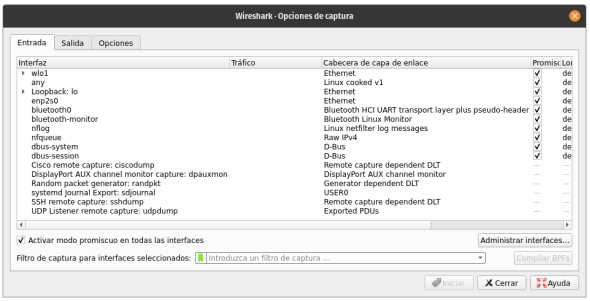


Imagen 76. Configuración de interfaces de Wireshark

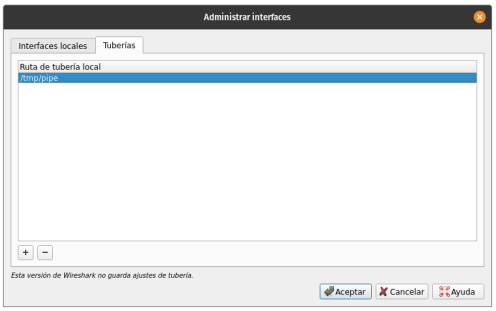


Imagen 77. Agregar el pipe creado

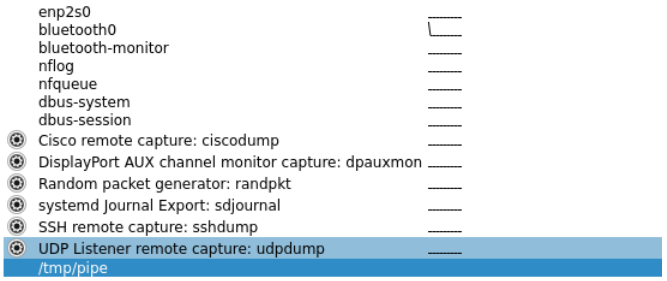


Imagen 78. Resultado de agregar el fichero Pipe

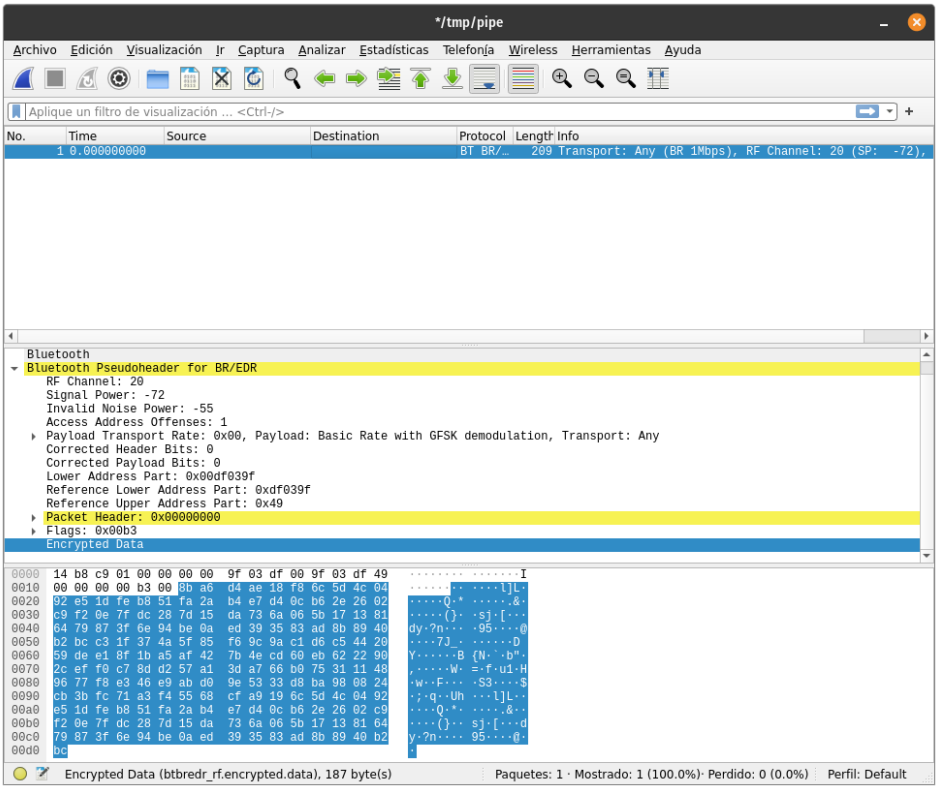


Imagen 79. Paquete capturado con datos encriptados

3.5 Programa central

El programa central consta de dos funciones: la primera sirve para detectar los diferentes dispositivos Bluetooth cercanos, mientras que la segunda sirve para facilitar una dirección MAC y realizar la detección de paquetes mediante ***Ubertooth.*** Podremos ver en la *Imagen 80* el menú del programa central.

Detección de dispositivos

Para realizar la primera operación el programa se basa en el controlador interno del sistema: *HCI*. Con este controlador podremos acceder a nuestro dispositivo, modificar el estado del mismo y permitir la escucha de nuevos dispositivos.

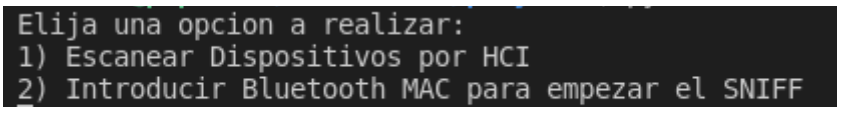


Imagen 80. Menú del programa central

Para esto se usan las librerías de *Pydbus*, que nos permiten un acceso al bus del sistema y *Bluez* que es el servicio que se encarga de la gestión de la información que pasa por el protocolo ***Bluetooth***. Dentro de este veremos que se puede acceder al controlador de cada dispositivo, las diferentes interfaces y las propiedades únicas de cada una de las mismas. El código que permite acceder a estos controladores se puede apreciar en la *Imagen 81*.

```
import pydbus
from typing import *
BUS_DEL_SISTEMA = pydbus.SystemBus() # obtenemos el bus del sistema
SERVICIO_BLUEZ = "org.bluez"      ## el servicio sobre el que se basara la aplic
CONTROLADOR_DE_OBJETOS= "org.freedesktop.DBus.ObjectManager"    ## El controlado
INTERFAZ_DE_ADAPTADOR="{}.Adapter1".format(SERVICIO_BLUEZ)      ## Obtenemos el
INTERFAZ_DE_DISPOSITIVO="{}.Device1".format(SERVICIO_BLUEZ)
GATT_SERVICE_INTERFACE = "{}.GattService1".format(SERVICIO_BLUEZ)
GATT_DESCRIPTOR_INTERFACE = "{}.GattDescriptor1".format(SERVICIO_BLUEZ)
GATT_CHARACTERISTIC_INTERFACE = "{}.GattCharacteristic1".format(SERVICIO_BLUEZ)
PROPERTIES_INTERFACE = "org.freedesktop.DBus.Properties"
class BlueZDBusException(Exception):
    pass
```

Imagen 81. Direcciones de interfaces, dispositivos y servicios dentro del sistema

Dentro de la implementación destacan los métodos *GetObjectManager* y *GetAdaptadorBus (ver Imagen 82)*. El primero, como su nombre dice, nos facilita un objeto con el listado completo de los diferentes adaptadores bluetooth del sistema que estén emparejados, todo esto en el formato *JSON*.

Mientras que el segundo se encarga de iterar sobre este objeto en búsqueda de un adaptador concreto dentro del bus del sistema, indicando que este existe y nos permite retornar las diferentes propiedades de este.

```
def GetObjectManager():
  controlador = BUS_DEL_SISTEMA.get(SERVICIO_BLUEZ, "/")[CONTROLADOR_DE_OBJETOS]
  return controlador.GetManagedObjects()


def GetAdaptadorBus(pattern=None):
  objetos = GetObjectManager()
  for path, ifaces in objetos.items():
    adaptador = ifaces.get(INTERFAZ_DE_ADAPTADOR)
    if adaptador is not None:
      if pattern is None:
        return BUS_DEL_SISTEMA.get(SERVICIO_BLUEZ,path)[INTERFAZ_DE_ADAPTADOR]
  else:
    raise BlueZDBusException("Adaptador bluetooth no encontrado")
```

Imagen 82. Métodos GetObjectManager y GetAdaptadorBus

Lo siguiente que nos encontramos son los métodos *GetAdaptador,* el cual nos permitirá obtener el adaptador y *GetDispositivos* que nos facilitará los diferentes dispositivos emparejados registrados. Ambos métodos son visibles en la *Imagen 83.*

```
def GetAdaptador(identificador=None):
  objetos = GetObjectManager()
  for path,ifaces in objetos.items():
    if path == identificador:
      adaptador = ifaces.get(INTERFAZ_DE_ADAPTADOR)
      if identificador is not None:
        return adaptador
  else:
    raise BlueZDBusException("Adaptador bluetooth no encontrado")

def GetDispositivos():
  objetos = GetObjectManager()
  for path, ifaces in objetos.items():
    if INTERFAZ_DE_DISPOSITIVO in ifaces.keys():
      yield(path,ifaces[GATT_SERVICE_INTERFACE])
```

Imagen 83. Métodos GetAdaptador y GetDispositivos

Como podemos ver en las *Imágenes 84 y 85*, el programa nos permite seleccionar los diferentes adaptadores dentro del sistema y, una vez elegido, nos mostrará el conjunto de dispositivos disponibles dentro del rango de nuestro adaptador. Dispondremos del alias del dispositivo, que es un nombre que se puede editar, la dirección MAC del dispositivo y la distancia del dispositivo.

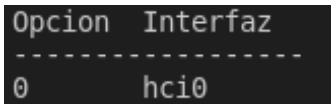


Imagen 84. Menú de los adaptadores Bluetooth del sistema

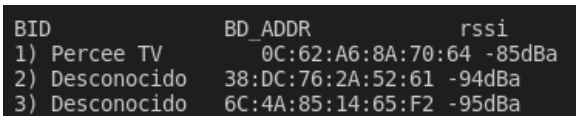


Imagen 85. Lista de los adaptadores Bluetooth detectados

Atacando paquetes con sniffer

Para este apartado usamos el dispositivo ***Ubertooth*** y ***Pyshar*** [14]. El primer paso es tener nuestro dispositivo Ubertooth conectado al sistema, de modo que nos pedirá la dirección MAC del dispositivo que estamos interesados en atacar.

Una vez el usuario ha introducido la dirección, se verifica que la información facilitada tiene un formato compatible con el de las direcciones MAC. A continuación, se dividirá la dirección en *LAP* y *UAP*.

Cuando se tenga esta información, ejecutaremos un *bash script* que nos servirá para poner el dispositivo Ubertooth en funcionamiento y que escuche todos los paquetes que sean transmitidos sobre Bluetooth clásico. En las *Imágenes 86 y 87* podremos ver el comando que ejecutan y la forma en la que manejan los parámetros, respectivamente y, posteriormente, el código *Python* que ejecuta el *bash script* en la *Imagen 88*.

```
set -- "${POSITIONAL_ARGS[@]}" # restore positional parameters

echo "LAP  = ${LAP}"
echo "UAP  = ${UAP}"
echo "LOGFILE   = ${LOG}"
echo "PCAP FILE = ${FILENAME}"
echo "TIME = ${TIME}"



ubertooth-rx -q ${FILENAME} -l ${LAP} -u ${UAP} -t ${TIME} > ${LOG}
```

Imagen 86. Registro de los diferentes parámetros del comando ubertooth-rx

```
POSITIONAL_ARGS=()

while [[ $# -gt 0 ]]; do
  case $1 in
    -l|--lap)
      LAP="$2"
      shift
      shift
      ;;
    -u|--UAP)
      UAP="$2"
      shift
      shift
      ;;
    -r|--register)
      LOG="$2"
      shift
      shift
      ;;
    -f|--filename)
      FILENAME="$2"
      shift
      shift
      ;;
    -t)
      TIME="$2"
      shift
      shift
      ;;
  esac
done
```

Imagen 87. Lector de los parámetros al script

Como podremos ver, este recibe diferentes parámetros:

- -l, para especificar el LAP del dispositivo que queremos escuchar.
- -u, para especificar el UAP del dispositivo que queremos escuchar.
- -r, para guardar los logs de la información que Ubertooth registra.
- -f, para especificar el fichero donde se guardarán los paquetes.
- -t, el tiempo que estará escuchando y capturando paquetes.

```
FILE_PCAP = 'informacion.pcap'
SCRIPT = "iniciar.sh"
LOG = 'registros.log'

def ejecutarComando(valores):
  os.system('bash {} -u {} -l {} -f {} -t {} -r {}'.format(SCRIPT,valores[0],valores[1],FILE_PCAP,60,LOG))
```

Imagen 88. Ejecución del script bash

Por defecto se correrá el comando durante 1 minuto y veremos que guardará los paquetes en el fichero "*informacion.pcap*" y los logs estarán dentro de "*registros.log*". El formato de los logs guardados se puede ver en la *Imagen 89*.

```
systime=1644435139 ch=60 LAP=df039f err=0 clkn=79948 clk_offset=2245 s=-40 n=-55 snr=15
systime=1644435139 ch=13 LAP=df039f err=0 clkn=80009 clk_offset=5361 s=-52 n=-55 snr=3
systime=1644435140 ch=13 LAP=df039f err=0 clkn=80041 clk_offset=5360 s=-57 n=-55 snr=-2
```

Imagen 89. Registros obtenidos

En dicha imagen se puede ver información tal como el *LAP*, el tiempo del sistema de cuando se registró, el *clkn* (valor del reloj del adaptador), la potencia de la señal y el ruido, entre otros. Una vez que ha pasado el tiempo establecido de captura, se obtendrá el listado completo de registro para posteriormente abrir el fichero *pcap* de los paquetes (ver *Imagen 90*):

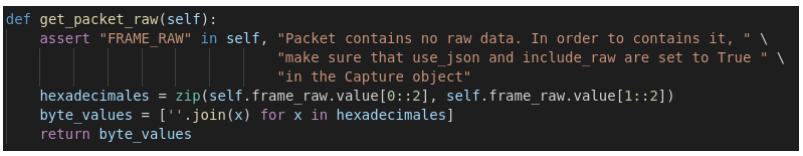
```
Data = []
RegistroLog = leerLog()
# Abrimos el fichero .pcap que es el que contiene la informacion relativa a los paquetes capturados
with open(FILE_PCAP) as pcap:
  capture = PipeCapture(pipe=pcap,use_json=True,include_raw=True)
  capture.apply_on_packets(registro_callback)
```

Imagen 90. Guardado de los paquetes

En el momento de abrir el fichero *pcap* y detectar un paquete, se obtendrá la información en formato de bytes, mediante un método modificado dentro de la librería y se extraerá la información que coincide con la que vimos dentro del *log* para buscar y encontrar los registros que corresponden al paquete (*ver Imágenes 91 a 94*).

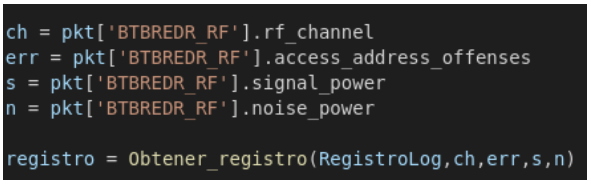
```
def registro_callback(pkt):
  sub_data=[]
  ch =0
  err=0
  s=0
  n=0
  vacio = True
  datos= ""
  datos = obtener_raw_data(pkt.get_packet_raw())
```

Imagen 91. Callback que analiza los paquetes con datos



```
def get_packet_raw(self):
    assert "FRAME_RAW" in self, "Packet contains no raw data. In order to contains it, " \
                                "make sure that use_json and include_raw are set to True " \
                                "in the Capture object"
    hexadecimales = zip(self.frame_raw.value[0::2], self.frame_raw.value[1::2])
    byte_values = [''.join(x) for x in hexadecimales]
    return byte_values
```

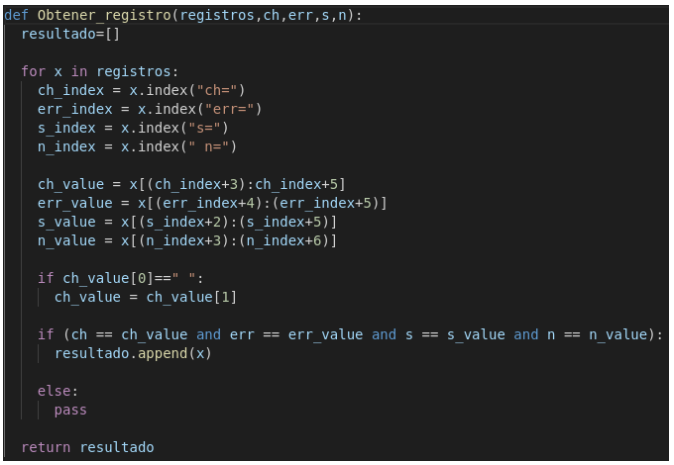
Imagen 92. Modificación de la librería de Pyshark



```
ch = pkt['BTBREDR_RF'].rf_channel
err = pkt['BTBREDR_RF'].access_address_offenses
s = pkt['BTBREDR_RF'].signal_power
n = pkt['BTBREDR_RF'].noise_power

registro = Obtener_registro(RegistroLog,ch,err,s,n)
```

Imagen 93. División de los datos



```
def Obtener_registro(registros,ch,err,s,n):
  resultado=[]

  for x in registros:
    ch_index = x.index("ch=")
    err_index = x.index("err=")
    s_index = x.index("s=")
    n_index = x.index(" n=")

    ch_value = x[(ch_index+3):ch_index+5]
    err_value = x[(err_index+4):(err_index+5)]
    s_value = x[(s_index+2):(s_index+5)]
    n_value = x[(n_index+3):(n_index+6)]

    if ch_value[0]==" ":
      ch_value = ch_value[1]

    if (ch == ch_value and err == err_value and s == s_value and n == n_value):
      resultado.append(x)

    else:
      pass

  return resultado
```

Imagen 94. Método que obtiene los registros con datos de cada paquete

Por último, verificamos si el paquete registrado tiene datos y, en caso de tenerlos, si son corruptos. Si ocurre esto desechamos el paquete. En caso contrario, lo guardamos para poder facilitar la información a la red neuronal y que esta nos dé el resultado final, puesto que esta necesita el reloj del adaptador maestro de la comunicación. Esto último se hace gracias al código mostrado en la *Imagen 95.*

```
def get_clkn(registro):
  clkn_index = registro.index("clkn=")
  clk_offset_index = registro.index("clk_offset=")
  clkn_value = registro[(clkn_index+5):(clk_offset_index-1)]
  return clkn_value
```

Imagen 95. Método que obtiene el valor del reloj

4. Resultados del proyecto

Una vez hemos creado los diferentes diccionarios y entrenado la red neuronal, los resultados los obtenemos siguiendo los pasos que se detallan a continuación:

Lo primero es tener el chat ya abierto y conectado, mientras que por otro lado tendremos el programa central activo. Ejecutamos el programa en modo Sniff y en el momento en que esté escuchando escribiremos dentro del chat. La secuencia de ejecución se puede ver desde la *Imagen 96* a la *98*.

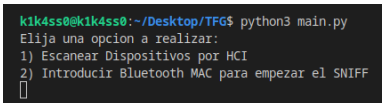


Imagen 96. Menú del programa principal

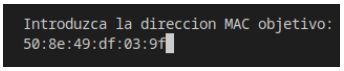


Imagen 97. Solicitud de la dirección MAC

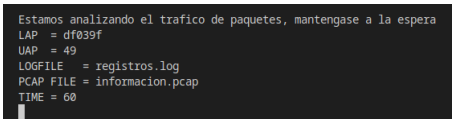


Imagen 98. Proceso de sniff de los paquetes del dispositivo

Como se puede observar en la *Imagen 98*, se guardan los *logs* de los paquetes recibidos por *Ubertooth* en el fichero "*registros.log*", mientras que la información de los propios paquetes capturados y analizados por *Pyshark* estarán en "*informacion.pcap".* Por defecto, se espera un tiempo de 60 segundos mientras se captura toda la información.

Como ya tenemos el programa escuchando los diferentes paquetes que tienen como emisor la MAC especificada, podemos enviar un mensaje en la aplicación (ver *Imágenes 99 y 100*).

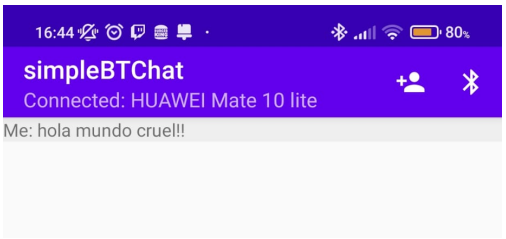


Imagen 99. Mensaje enviado desde el dispositivo emisor

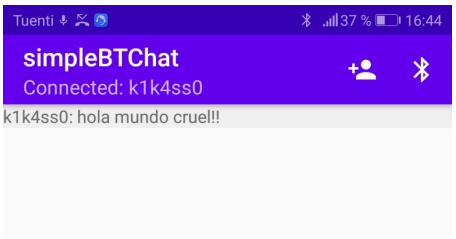


Imagen 100. Mensaje recibido por el dispositivo receptor

Una vez pasado el tiempo podremos observar los resultados en la *Imagen 101*.

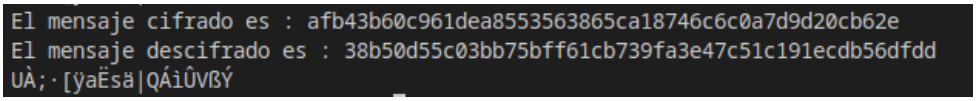


Imagen 101. Resultado del descifrado de los datos del paquete capturado

Como podemos observar claramente, los caracteres no coinciden con los del mensaje enviado.

5. Conclusiones

Después de realizar diferentes pruebas modificando los diferentes tipos de modelos desarrollados, así como la gran cantidad de muestras facilitadas para el entrenamiento, se puede concluir que el cifrado del Bluetooth clásico no es vulnerable ante un ataque basado en Deep Learning.

Estas conclusiones no solo son observables a partir de los resultados finales del proyecto, sino que incluso en el momento de realizar los entrenamientos de todos los modelos, se encontró una gran dificultad en los intentos de la red neuronal por abstraer el problema y aprender en base a las muestras utilizadas.

Independientemente del modelo utilizado, el grado de error siempre variaba entre un 63% y un 67% de diferencia con respecto al resultado esperado. Además, no solo este grado de error disminuye de una forma poco apreciable al utilizar más datos, sino que llegado a un determinado punto, hasta llegaba a aumentar.

6. Conclusions

After multiple tests and changes between different model types, as well as the amount of samples that had been available to train those models, we can conclude that classic Bluetooth cipher is not vulnerable to a DL based attack.

These conclusions are not even apreciable by the final results, it is also, at the training moment while all models had been tested, we could notice strong difficulties with multiple tries to make the neural network abstract the posed problem and learn from generated samples.

Taking the used model aside, loss values always seemed to oscillate between 63% and 67% regardless of the expected result, even more, this loss not only decreased in a insignificant way, but this loss got higher at a certain point.

7. Presupuesto

7.1 Personal

En lo que se refiere a costos de personal, computando en concepto de tiempo, se puede dividir en lo siguiente:

Tarea	Tiempo empleado (Horas)
Investigación y comprensión del protocolo Bluetooth	100
Implementación de programa para leer información de los paquetes Bluetooth	40
Implementación del Protocolo E0	40
Implementación del Algoritmo de Machine Learning	100
Implementación de la aplicación Android	25
Creación del Data Set	6
Tiempo total de entrenamientos de la red neuronal	50
Redacción del informe	15

7.2 Componentes

Para la implementación del trabajo se ha necesitado de los siguientes dispositivos hardware:

Elemento	Costo (Euros)
Raspberry Pi 3 Model B	60
Tarjeta SD para instalación del SO Raspberry Pi	4.50
Disco duro Seagate 2TB	60
nRF52840	5.78
Ubertooth	46.68

7.3 Salarios estimados

Se incluye el salario estimado para los profesionales especializados en los diferentes campos relacionados con el proyecto:

Profesión	Salario anual	Precio/hora	Sueldo por participación en proyecto
Ingeniero de criptografía	106.800	37	6.660
Ingeniero de Deep Learning	41.858	14.50	2.262
Desarrollador móvil	32.165	11.17	279
Total			9.201

7.4 Coste total

Para la finalización del trabajo se ha obtenido la siguiente inversión de costos y tiempo empleados para el proyecto:

Total tiempo(horas)	Total inversión económica (euros)	Costo sueldo profesional	Total
376	177	9.201	9.378

8. Bibliografía

1. Padgette, J., Scarfone, K., Chen, L. Guide to bluetooth security. *NIST Special Publication*, *800*, 121. 2017.
2. Vainio, J. T. Bluetooth security. *Proceedings of Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory, Seminar on Internetworking: Ad Hoc Networking, Spring* (Vol. 5). 2000.
3. Artículo sobre el TDD (Time Division Duplex)
   https://www.cablefree.net/wirelesstechnology/fdd-vs-tdd/
4. Documentación oficial de Bluetooth para la versión 5.2
   https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=478726
5. Singelée, D., Preneel, B. Security overview of Bluetooth. *Technical Report COSIC*. 2004.
6. Artículo sobre la diferencia entre Machine Learning y Deep Learning
   https://flatironschool.com/blog/deep-learning-vs-machine-learning/
7. Configuración de capas de Deep Learning.
   https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3#:~:text=The%20number%20of%20hidden%20neurons,size%20of%20the%20input%20layer
8. Estándares de número de capas según los problemas planteados
   https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
9. Documentación oficial de Pytorch
   https://pytorch.org/docs/stable/index.html
10. Tipos de datos de Pytorch
    https://pytorch.org/docs/stable/tensors.html
11. Guía sobre cómo utilizar el comando Ubertooth-rx
    https://manpages.debian.org/experimental/ubertooth/ubertooth-rx.1.en.html
12. Repositorio con las librerías y herramientas del proyecto Ubertooth
    https://github.com/greatscottgadgets/ubertooth
13. Documentación oficial de Bluetooth para Android en la versión de Java
    https://developer.android.com/guide/topics/connectivity/bluetooth?hl=es-419
14. Repositorio del proyecto Pyshark
    https://github.com/KimiNewt/pyshark