Universidad de Valladolid

Escuela Técnica Superior de
Ingenieros de Telecomunicación

TRABAJO FIN DE MÁSTER

Máster en Ingeniería de Telecomunicación

Identificación de tarjetas mediante radio definida por software

Autor:

D. Samuel Adrián Peña

Tutor:

Dr. Juan Carlos García Escartín

Valladolid, 26 de septiembre de 2025

DESCRIPCIÓN DEL TRABAJO FIN DE GRADO

TITULO: Identificación de tarjetas mediante radio definida por software
AUTOR: D. Samuel Adrián Peña
TUTOR: Dr. Juan Carlos García Escartín
DEPARTAMENTO: Teoría de la Señal, Comunicaciones e Ingeniería Telemática

MIEMBROS DEL TRIBUNAL

PRESIDENTE: Dr. Pedro Chamorro Posada
VOCAL: Dra. María Jesús González Morales
SECRETARIO: Dr. Ramón de la Rosa Steinz
PRESIDENTE SUPLENTE: Manuel Rodríguez Cayetano
VOCAL SUPLENTE: Luis Miguel San José Revuelta
SECRETARIO SUPLENTE: Miriam Antón Rodríguez
FECHA: 26 de septiembre de 2025
CALIFICACIÓN:

Resumen

Este trabajo pretende analizar la viabilidad de uso de la frecuencia de resonancia de un sistema NFC como huella digital. Cada tarjeta NFC presenta una frecuencia diferente debido a los procesos propios de fabricación y difícilmente clonables. El proyecto utiliza materiales de bajo coste como los populares analizadores de redes vectoriales nanoVNA que, en conjunto con un programa que permita adquirir y analizar los datos obtenidos, puedan detectar mediante la frecuencia de resonancia del sistema la tarjeta a autenticar. Con el nanoVNA obtenemos el valor mínimo de la relación de onda estacionaria, que equivale al mejor acoplamiento del sistema LC de la tarjeta NFC y el propio instrumental. El experimento se ha dado durante varios meses y en diferentes lugares para abordar si los cambios de tiempo, lugar, clima y temperatura afectan en gran medida en el sistema, siendo bastante invariante en el tiempo. Tras el análisis de los resultados, el sistema es capaz de detectar con éxito la mitad de los ensayos, lo que indica que es posible utilizar este sistema como mecanismo de detección, pero no suficiente con el estudio para ponerlo en producción. Los datos obtenidos son una buena base para continuar la investigación en un futuro y utilizar otro tipo de instrumentación y o mecanismos para mejorar la tasa de éxito del sistema.

Palabras clave: NFC, nanoVNA, huella digital, parámetros S, SWR, PUF, detección, autenticación, resonancia.

Abstract

This research analyses the potential of using the resonant frequency of NFC systems as a fingerprint. Each NFC card has a different frequency due to variations in the manufacturing processes and it can't be cloned easily. The project uses low-cost material such as nanoVNA, an economical vectorial network analyser, together with a PC application, to acquire and analyse the data obtained from the card in order to authenticate it using its resonant frequency. NanoVNA can get the minimum value of the standing wave ratio. This is the equivalent to the best coupling of the LC circuit of the NFC card and analyser considered together. The experiment has been done for several months in different places with different weather and temperature conditions. We can check if these and other variables affect the system's resonance or not. After analysing the results, the system is able to detect at least the half of cards. The results indicate that the system can be used as a detection mechanism, but the research is not advanced enough to be used in production. The obtained data give a good foundation to continue the research in the future using other kind of analysers or mechanisms to improve the success rate of the system.

Keywords: NFC, nanoVNA, fingerprint, S parameters, SWR, PUF, detection, authentication, resonance

A mis padres por el apoyo y ayuda que he han dado en todo momento, así como todos los recursos que me han facilitado para hacer todo más fácil.

A mi hermano Noé por ser mi pilar de apoyo fundamental, por estar ahí en cada momento ayudando y dando todo su apoyo en cada momento.

A Kira por insistirme y estar ahí cada día recordándome que debía terminar este trabajo y por la ayuda recibida.

A Juan Carlos, por se el tutor de este proyecto, al cual doy las gracias y agradezco todo el trabajo de guiado y tutorización recibido. Por estar detrás de mí para que este proyecto llegara a su fin.

Al resto de mi familia, por apoyarme en todos mis propósitos y proyectos, y en ocasiones darme el empujón que necesitaba para ponerlo en práctica.

A todos aquellos que durante este tiempo me han ayudado, aportado y apoyado en mis proyectos y propósitos, tanto profesionales como personales.

Índice de Contenidos

1 Objetivos

La seguridad y la suplantación de identidad es un tema que preocupa hoy en día en gran medida a nuestra sociedad. Los investigadores trabajan por investigar y descubrir nuevas técnicas de seguridad y en reforzar las existentes para evitar que usuarios mal intencionados pongan en riego los sistemas y puedan hacerse pasar por usuarios que realmente no lo son. Estos utilizan diversas técnicas como el clonado de credenciales, acceso ilegítimo a bases de datos y otras técnicas que permitan conseguir el fin buscado.

En este proyecto se busca investigar sobre un mecanismo de autenticación de usuarios no clonable, basado en Physically Unclonable Functions (PUF). Esta técnica consiste en añadir elementos durante la construcción de manera controlada que no se puedan clonar, o como el caso que se va a estudiar en este trabajo, elementos aleatorios de valores de componentes por las tolerancias de producción en la construcción de los componentes. El estudio se realizará sobre tarjetas o tags NFC, ya que por sí solas ya poseen un mecanismo de seguridad integrado y, por otra parte, las tolerancias de la bobina que actúa como antena y el condensador de acoplo del sistema LC poseen valores "aleatorios" en su construcción debido a las tolerancias de valores. En concreto se pretende utilizar la frecuencia de resonancia del sistema resultante, tarjeta y sistema de medida, como huella digital que permita detectar de forma única a cada una de las tarjetas del sistema.

Para el desarrollo del trabajo se ha contemplado el uso de un dispositivo instrumental de análisis, nanoVNA, que es un analizador de redes vectorial de bajo coste y una antena diseñada para aplicaciones NFC, en la sección 4 se explica que es el nanoVNA y los parámetros S con los que trabaja este tipo de instrumentación. Mediante los datos obtenidos de parámetros S, se calculará la relación de onda estacionaria (SWR) para obtener la frecuencia de resonancia del sistema completo acoplado.

Para llevar a cabo el estudio y análisis se ha diseñado un banco experimental de medidas, consistente en tomar una medida patrón de cada una de las tarjetas y posteriormente la realización de medidas para la detección, mediante comparación y sistema de umbrales, de cada una de las tarjetas y observar los resultados. En la sección 6 se detalla el diseño experimental a seguir para realizar los experimentos y una visión general de lo que será el estudio.

El análisis y procesado de datos se realiza mediante un programa creado al efecto en lenguaje C# y .Net Framework. Se ha utilizado este lenguaje y framework por la facilidad en su uso, en el dominio ya obtenido con este lenguaje y por la facilidad de migración a .Net multiplataforma para su uso en múltiples sistemas operativos. En la sección 7 se detallan las características y funciones de la aplicación de evaluación de tarjetas, consistente en funciones de adquisición de datos, base de datos de tarjetas patrón y

tarjetas detectadas, y funciones iterativas para detectar, bajo un umbral máximo, la tarjeta en estudio.

Una vez realizados los experimentos, se han obtenido unos resultados que podemos dar como satisfactorios en su conjunto, y para la mayoría de tarjetas también, aunque en algunos casos ha resultado imposible la detección de la tarjeta. Se comprueba que la detección de tarjetas, usando este método, es posible obteniendo hasta un 50% de éxito en detección, valor satisfactorio para nuestro estudio y que permite realizar nuevas investigaciones siguiendo la línea abierta en esta investigación a posibles mejoras. En la sección 8 se pueden ver en detalle la fase experimental y análisis.

Para concluir con el estudio, podemos afirmar que se cumple el objetivo de poder distinguir entre diferentes tarjetas utilizando el método de frecuencia de resonancia como huella digital, pero no se cumple el objetivo de detectar inequívocamente a todas las tarjetas utilizadas en el ensayo, que puede ser debido a varios factores limitantes como posibilidad de coincidencia de frecuencia de resonancia, frecuencias de resonancia demasiado cercanas que no es posible discriminar con nanoVNA, no uso de un banco de lecturas de la misma tarjeta, en lugar de una única lectura patrón, la colocación de la tarjeta sobre la antena, entre otras limitaciones. El estudio proporciona una base para investigar sobre esta técnica en investigaciones futuras, pudiendo continuar por el uso de sistemas estadísticos y analizando la varianza de cada tarjeta para poder decidir la tarjeta en estudio. En la sección 9 se pueden ver las conclusiones detalladas, así como las limitaciones del estudio y líneas futuras a seguir.

2 Introducción

El continuo avance de las tecnologías inalámbricas hace que se desarrollen métodos de comunicación fiables y sencillos para los usuarios. En la actualidad múltiples tecnologías inalámbricas han sustituido a las conexiones cableadas. Cada tecnología nos ofrece diferentes características como velocidad de transmisión, consumo de energía o métodos de seguridad [1]. La tecnología Near Field Communication (NFC) es una tecnología de comunicación inalámbrica que hace uso del campo magnético para realizar la transmisión y recepción de información en corto alcance. El intercambio de información se realiza a alta velocidad en un alcance máximo de 20 centímetros, proporcionando en primera instancia un alto grado de seguridad debido al corto alcance de las comunicaciones [2]. Esta tecnología está presente en múltiples dispositivos y medios como *smartphones*, lectores NFC, etiquetas y *tags*. Entre los usos actuales de esta tecnología se encuentran medios de pago electrónico (tarjetas de crédito), configuración de dispositivos móviles e IoT, almacenamiento e intercambio de información (número de teléfono, datos de contacto…), identificación o control de acceso entre otros [3] [4].

Los *tags* NFC básicos, del tipo MIFARE Classic EV1, se componen de un número de identificación único (UID) de 7 bytes y un espacio de memoria para almacenar información variable en una EEPROM [5]. La seguridad de los *tags* NFC viene proporcionada por varios factores, como el corto alcance de comunicación o el cifrado que proporciona los MIFARE Plus SE [6]. También se pueden utilizar mecanismos de cifrado externo para añadir una capa más de seguridad.

Aplicaciones como control de acceso para apertura de puertas, sistemas de registro horario u otros sistemas similares utilizan *tags* NFC para identificar a los usuarios. Estas aplicaciones habitualmente utilizan el UID único de cada tarjeta para validar al usuario, sin necesidad de utilizar complejos sistemas de cifrado, ni utilizar *tags* de mayor costo como puede ser MIFARE Plus SE utilizadas en otras aplicaciones más sensibles como tarjetas monedero (necesidad de almacenamiento de un saldo). Hasta aquí con los datos expuestos, y de forma ideal, puede parecer que la comunicación de corto alcance y el UID grabado en el *tag* son suficientes para identificar al usuario ante el sistema.

Profundizando en la tecnología, se puede observar que existe un modo de operación "Card Emulation mode" que permite operar como una tarjeta sin contacto y ser leída por un lector NFC, con el propósito de emular una tarjeta de crédito u otro sistema de *tickets* [4]. Por otra parte, existen tarjetas compatibles con MIFARE Classic que poseen la característica de tener "UID modificable", haciendo que existan múltiples tarjetas con mismo UID, además de utilizar un smartphone pasa emular/clonar un tag NFC y que desde hace tiempo la seguridad de MIFARE ha sido comprometida [7].

Analizando los casos de seguridad anteriores, podemos afirmar que en prácticamente todos los casos la seguridad se basa en sistemas software, salvo en la comunicación

cercana que es una característica física. Con ello sacamos la conclusión de seguridad pobre, pudiendo clonar identificadores de *tags* o descifrar los datos mediante mecanismos de fuerza bruta, dejando los sistemas expuestos.

Por ello es necesario utilizar un sistema no clonable para identificar a los usuarios objeto de investigación en este estudio mediante uso de parámetros físicos de fabricación debido a pequeños desajustes con los componentes del sistema, siendo prácticamente poco probable obtener *tags* idénticos físicamente [8].

Se analizará la frecuencia de resonancia de cada *tag*, junto con una antena de lectura que se utilizará durante todo el estudio, para determinar la huella digital de cada *tag*, índice de repetibilidad y desviación de las diferentes frecuencias de resonancia. Realmente la frecuencia de resonancia obtenida no es la de cada *tag* de manera aislada, sino que será la del sistema en conjunto para el *tag*, antena, línea de transmisión y nanoVNA utilizados, siendo un modelo equivalente similar al de la Figura 1. Por simplicidad, en el resto del documento, nos referiremos a la frecuencia de resonancia del sistema completo simplemente como frecuencia de resonancia.

Para nuestros propósitos utilizaremos un dispositivo nanoVNA, analizador de redes vectorial de bajo coste y una resolución baja pero aceptable para determinar la frecuencia de resonancia de cada *tag* NFC. Con las conclusiones que obtengamos con el estudio se podrá determinar la variabilidad y la posibilidad de uso como un sistema funcional de identificación de usuarios.

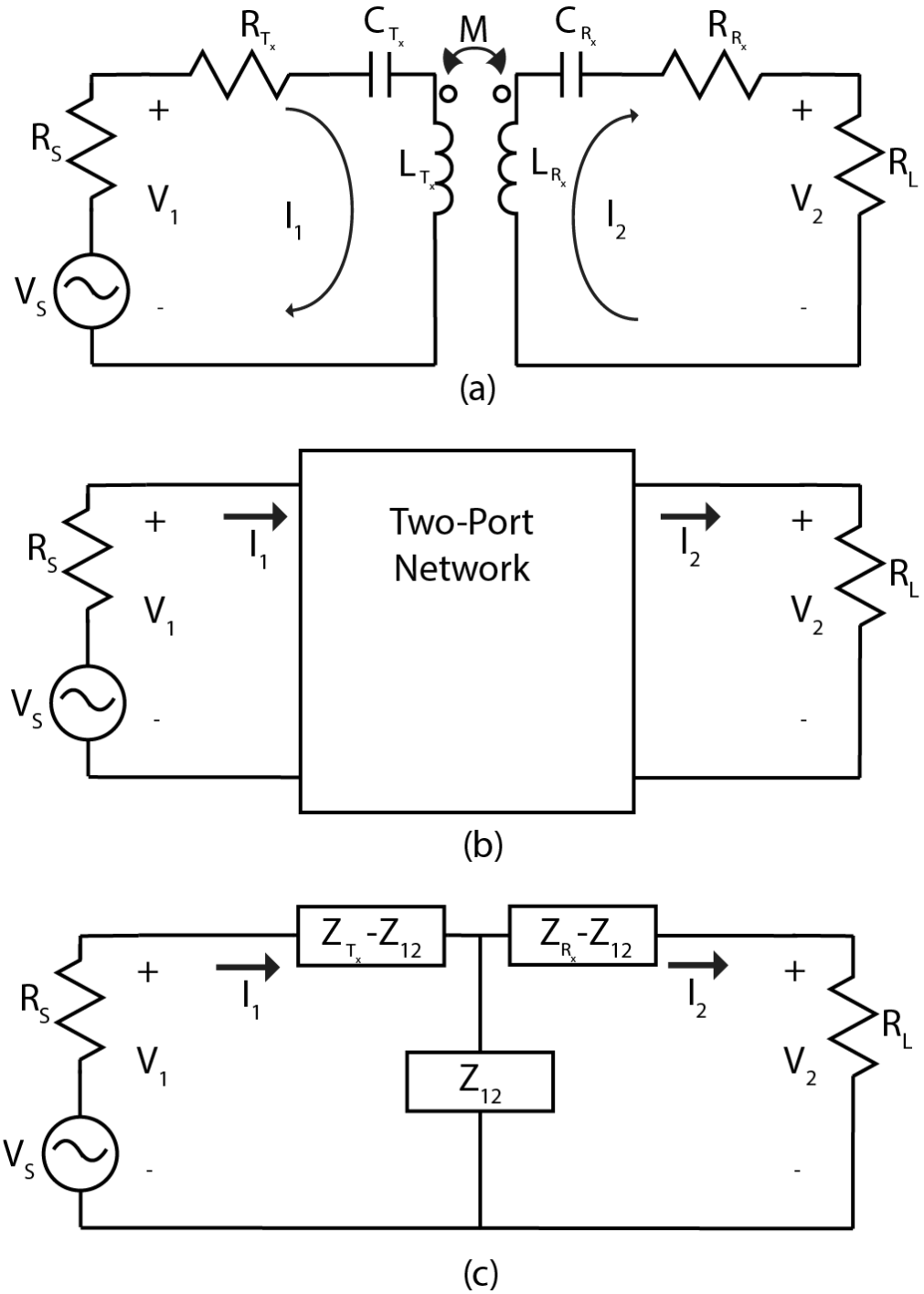

Figura 1. Red de dos puertos. (a) Sistema de dos bobinas con parámetro M de acoplamiento, (b) red general de dos puertos equivalente, dentro se encuentra el sistema de bobinas y capacitores acoplados mediante parámetro M, (c) red de dos puertos T-equivalente utilizando parámetros Z, convirtiendo en impedancias equivalente el sistema (a) [9].

2.1 Uso de la frecuencia de resonancia como huella digital

El objeto de este estudio tiene como objetivo validar la frecuencia de resonancia como huella digital de *tags* NFC. La frecuencia de resonancia de un sistema simple se describe según la ecuación 1, siendo L el valor de la inductancia y C el valor del capacitor [10]:

$$f_r = \frac{1}{2\pi\sqrt{LC}}. \quad (1)$$

En un entorno ideal, el valor de la frecuencia de resonancia obtenida con la ecuación 1 sería siempre invariable si se utilizan siempre componentes de mismos valores para L y C. En un entorno real, existen factores que alteran la frecuencia de resonancia real del sistema. Esto es debido a los valores de L y C, ya que estos componentes, debido a su proceso de fabricación, han tenido alteraciones del valor nominal de su inductancia y capacidad. Una mínima variación en L y/o C alterará al menos mínimamente la frecuencia de resonancia lo suficiente para determinar que el sistema no trabaja perfectamente en la frecuencia de resonancia de diseño o ideal. Las variaciones de estos componentes sobre su valor nominal se denomina tolerancia del componente y suele venir dado por porcentajes. Estas variaciones vienen dadas por su fabricación, variaciones en el material utilizado u otros factores. También puede afectar al sistema resonante otros factores como la temperatura o humedad [10]. En fase de diseño, de cada sistema, los desarrolladores y diseñadores tienen en cuenta este tipo de tolerancias y, salvo en sistemas muy críticos que se pueden añadir circuitos de compensación, estas variaciones no afectan al correcto funcionamiento de los sistemas.

En el caso de NFC, podemos modelarlo como un sistema simple resonante LC, ya que se compone de una bobina que actúa como antena y un capacitor de acoplamiento al sistema resonante. La frecuencia de resonancia nominal del sistema NFC es de 13,56MHz [11] y las variaciones en los *tags* de la frecuencia de resonancia nominal no son críticos para el correcto funcionamiento del sistema. Es cierto que, para el funcionamiento óptimo del sistema, el perfecto acoplamiento es la mejor opción. Para la validación de usuarios un acoplamiento menos preciso es válido para el propósito de obtener los pocos datos almacenados del usuario.

Como claro ejemplo de variación de la frecuencia de resonancia de dos *tags* NFC distintos, se ha realizado una prueba con dos etiquetas al azar y, se ha analizado su espectro y visualizado la frecuencia de resonancia para cada una de ellas (valor mínimo de SWR).

Para el primer caso, que se puede ver en la Figura 2, se observa que este *tag* tiene una frecuencia de resonancia de 15,86 MHZ. Realizando el mismo análisis de espectro para el segundo *tag*, según la Figura 3, se observa que la frecuencia de resonancia es de 15,68 MHz.

Con los datos anteriores, podemos afirmar que entre ambos *tags* hay una desviación de 0,18 MHz entre *tags*, y una desviación de más de 2 MHz entre los *tags* y la frecuencia de resonancia nominal del sistema NFC.

Suponiendo que para cada *tag* se obtiene una frecuencia de resonancia diferente, que esta viene dada por factores de fabricación aleatorios y que previamente no se sabe con incertidumbre que frecuencia de resonancia tendrá cada tarjeta previamente al proceso de fabricación, podemos utilizar esta característica como elemento único e invariable para cada tarjeta como si de una huella digital se tratase.

Por ello es viable la realización de un estudio de detección de tarjetas basado en la frecuencia de resonancia, debido a su aleatoriedad en el momento de la fabricación y se puede usar la tarjeta como una función física no clonable (PUF) [8].

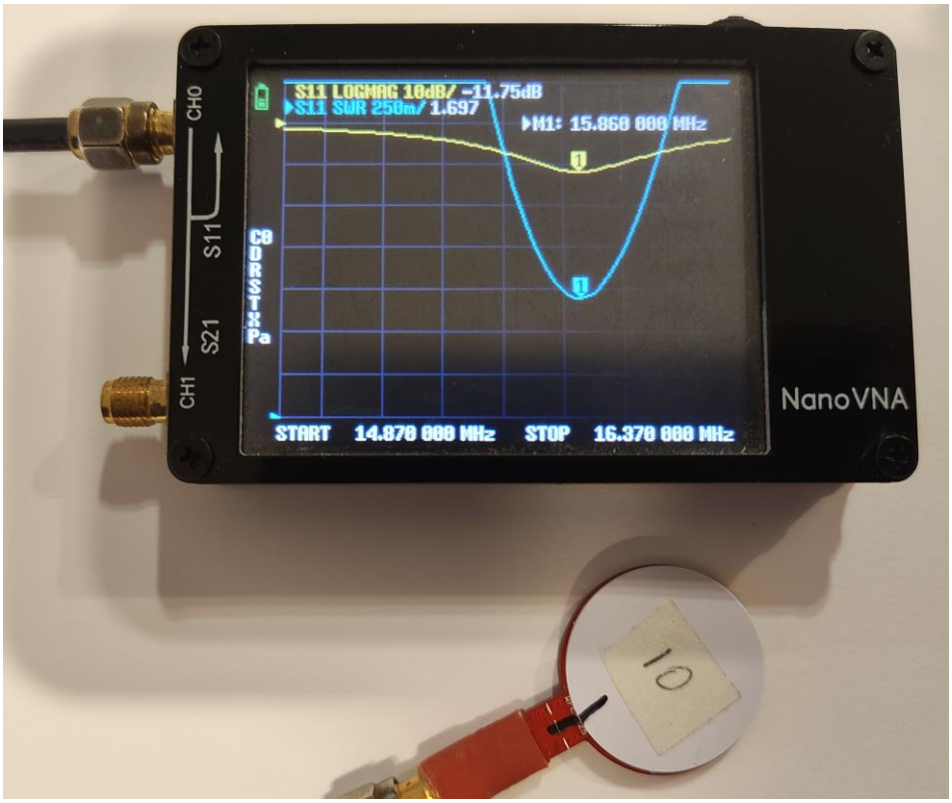

Figura 2. Valores de SWR espectral para el *tag* 10, con frecuencia de resonancia 15,86 MHz.

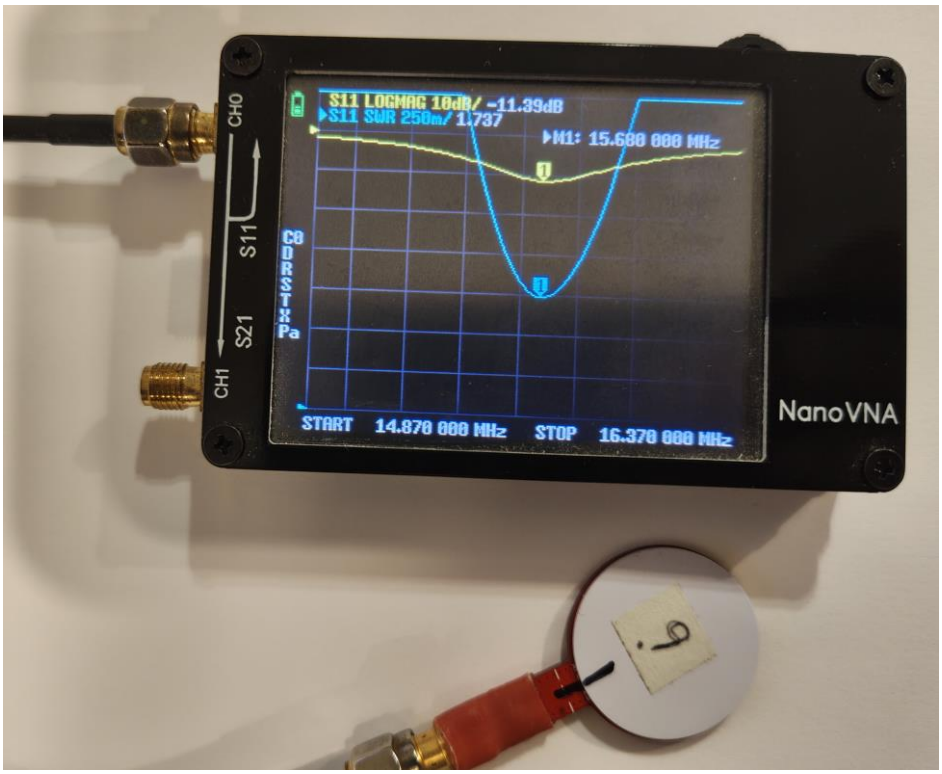

Figura 3. Valores de SWR espectral para el *tag* 9, con frecuencia de resonancia 15,68 MHz.

3 Near Field Communication

Near Field Communication (NFC) es una tecnología de comunicación que lleva investigándose desde mediados de los años 90. Los investigadores marcaron las directrices y características de una tecnología de comunicación a muy corto alcance, alrededor de 20 cm, que permitiría el intercambio de información entre dispositivos [12]. Para ello se analizaron dos formas físicas de conseguir el objetivo, la primera mediante códigos QR, actualmente muy extendidos en la sociedad, y la tecnología basada en identificación por Radio Frecuencia (RFID) mediante *tags* identificadores de recursos [12]. La primera es una opción fácil de implementar y de "coste cero" ya que únicamente es necesario imprimir el código en el lugar deseado, pero, por el contra, esto puede hacer que sea vulnerable, por la facilidad de cambio de códigos por usuarios malintencionados, o que sea antiestético. La segunda es la opción por la cual se encauzó NFC, utilizando la tecnología RFID para la lectura e intercambio de información mediante *tags*. Es un método más estético y menos vulnerable a la vista que el código QR, pero a su vez lo hace más costoso por el precio de los *tags* (pegatinas o tarjetas) [12].

Ya en 2004 se crea una organización con el objetivo de regularizar y trabajar en los estándares de la tecnología NFC. Este es el "NFC Forum" [4] [13]. Inicialmente la organización estaba compuesta por tres grandes fabricantes de tecnología, Phillips (actualmente NXP en la línea de semiconductores), Sony y Nokia, con el paso del tiempo se fueron uniendo otras empresas al grupo de trabajo de NFC Forum [13].

Esta tecnología emplea la banda de frecuencia de 13,56 MHz, de uso libre y sin licencia, para realizar la comunicación entre dispositivos y se regula por diferentes estándares de la Oficina Internacional de Estandarización (ISO) [5].

La regulación principal para Europa es ISO/IEC 14443 que define las características físicas, potencias a utilizar, sistema anticolisión y protocolos de comunicación [13] [14].

La norma ISO/IEC 14443 consta de las siguientes sub-normas:

1. ISO/IEC 14443-1 – *Physical characteristics*
2. ISO/IEC 14443-2 – *Radio frequency power and signal interface*
3. ISO/IEC 14443-3 – *Initialization and anticollision*
4. ISO/IEC 14443-4 – *Transmission protocol*

Otras normas ISO regulan otros aspectos de la tecnología NFC, como la ISO/IEC 15693 que regula la conformidad de las tarjetas, la ISO/IEC 18092 regula la conformidad de los dispositivos, u otros estándares no incluidos en ISO que regulan la conformidad de lectores y tarjetas capaces de comunicarse mediante NFC [4] [12].

3.1 Principales características de NFC

3.1.1 Corto alcance

La tecnología permite una comunicación de hasta 20 cm, y en la mayoría de casos esta se limita a un valor muy inferior de solo 2 cm. Esto implica tener los dispositivos que se comunicarán (lector y *tag*) prácticamente pegados entre sí para que la comunicación sea posible, proporcionando un nivel de seguridad adicional por el corto alcance [2] [5] [15].

3.1.2 Baja potencia de transmisión

La potencia de transmisión es unos pocos mW, pero permite una trasferencia de potencia para la carga de dispositivos de hasta 1W, y su uso se realiza en banda de frecuencia libre y sin licencia. Se pretende minimizar las interferencias con otros sistemas y usa una comunicación de campo cercano [15]. Por otra parte, la limitación de potencia es un tanto a favor por el bajo consumo del dispositivo que lo hace atractivo, sin necesidad de cargar con frecuencia las baterías del mismo [5].

3.1.3 Compatibilidad

La tecnología NFC es compatible con otras tecnologías. Un ejemplo es el de CARCONNECTIVITY Consortium que utiliza la tecnología NFC para desarrollar un sistema interoperable de llaves digitales, ambos organismos trabajan en conjunto para desarrollar esta solución de llaves digitales [15].

También se puede utilizar en conjunto con las tecnologías de comunicación inalámbricas Wi-Fi y Bluetooth [5] [15].

Dispositivos portátiles como *smartphones* y *wearables* son compatibles con la tecnología, empleando modos de lectura e incluso de emulación para su uso en aplicaciones diarias como tarjetas de crédito o control de accesos [12] [5].

3.1.4 Seguridad

NFC permite la integración de sistemas de seguridad mediante cifrado y otros sistemas *software* desarrollados por los usuarios. Las tarjetas MiFare proporcionan mecanismos de seguridad y cifrado que limitan el acceso y modificación de la información de usuarios malintencionados [16] [6].

La seguridad *software* en conjunto con el corto alcance hacen un sistema híbrido que mejora la seguridad con respecto a otros sistemas de comunicación [2] [5].

3.2 Componentes básicos de la Tecnología NFC

Según la entidad encargada de regular y estandarizar la tecnología NFC, existe tres componentes básicos en el ecosistema NFC:

- **Lectores NFC:** Dispositivos activos encargados de escribir y leer la información de otros dispositivos NFC o de etiquetas NFC [4] [5].
- **Etiquetas NFC (*tags*):** Son pequeños transpondedores, usualmente en forma de etiqueta o tarjeta. Son elementos pasivos y de pequeño tamaño. Permiten almacenar y transmitir datos entre etiqueta y lector [4] [5].
- **Dispositivos compatibles NFC:** Son dispositivos con funcionalidades más amplias que las limitadas al NFC. Pueden ser activos o pasivos y se pueden comportar como etiquetas o como lectores. Como ejemplos tenemos smartphones, wearables, medios de pago y otros dispositivos [4] [5]

3.3 Acoplamiento inductivo NFC

La tecnología NFC se basa en la comunicación de campo magnético cercano. El acoplamiento inductivo se aplica en NFC a los dispositivos transmisores y receptores [17]. En la Figura 4 se puede observar un esquema básico del sistema. El transmisor genera un campo magnético alterno en la bobina primaria que excita, según las leyes de Faraday, un campo magnético alterno secundario en la segunda bobina. Con este acoplamiento es posible la transferencia de potencia inalámbrica de un dispositivo a otro.

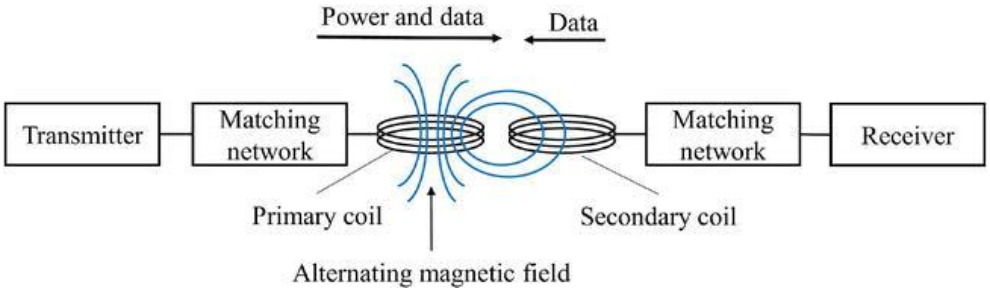

Figura 4. Acoplamiento inductivo entre bobinas de transmisor y receptor [17].

3.3.1 Sistema inductivo de bobina

El sistema se puede describir usando un modelo de autoinductancia mutua. Para ello vamos a utiliza una geometría de bobinas circulares o rectangulares [17] [9]. En la Figura 5 se representa un sistema de inducción magnética de dos bobinas. A partir de este modelo y geometría podemos obtener un sistema de ecuaciones de autoinductancia mutua.

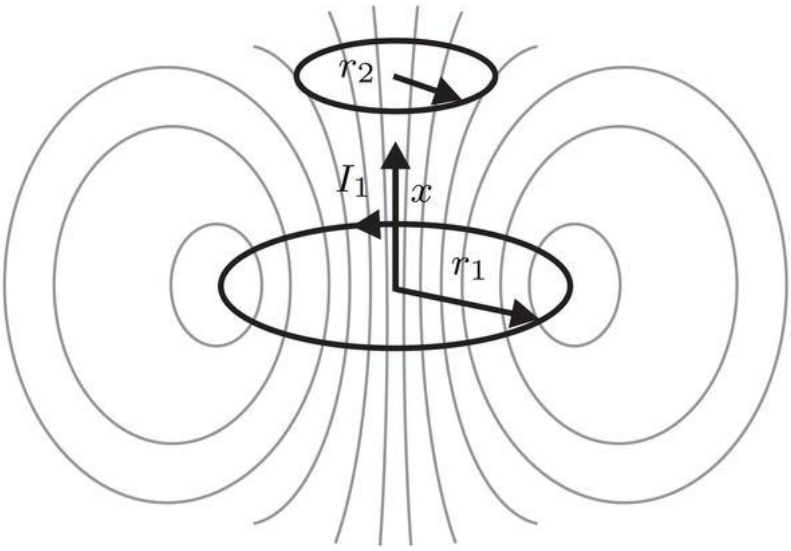

Figura 5. Sistema de inducción magnética de dos bobinas [17].

Para nuestros propósitos se van a utilizar las ecuaciones involucradas en el modelado y cálculo de las impedancias del transmisor y del receptor. Para ello necesitamos saber las ecuaciones de la inducción mutua entre dos bobinas (M), el coeficiente de acoplamiento (k) y las ecuaciones de impedancias (Zx) [17] [9]:

$$M = \frac{\mu\pi N_1 N_2 r_1^2 r_2^2}{2\sqrt{\left(r_1^2+x^2\right)^2}}. \tag{2}$$

La ecuación 2 expresa la inducción magnética entre dos bobinas, siendo μ el coeficiente de permeabilidad magnética, N_1 y N_2 el número de espiras de cada bobina, r_1 y r_2 son los radios de las bobinas, y x la separación entre bobinas.

La siguiente ecuación expresa el factor de acoplamiento entre dos bobinas (k):

$$k = \frac{M}{\sqrt{L_1 L_2}}, \tag{3}$$

siendo M la inducción magnética entre dos bobinas, calculada en la ecuación anterior y, L_1 y L_2 las inductancias de las dos bobinas, respectivamente.

Para esta explicación vamos a utilizar el modelado de redes utilizado en la Figura 1 para obtener las impedancias de la red de dos elementos en un sistema de dos bobinas y en un sistema equivalente que utiliza parámetros Z [9].

A partir de la Figura 1 (a) podemos obtener las ecuaciones de la impedancia del sistema transmisor (Z_{Tx}) y de la misma manera la impedancia del sistema receptor (Z_{Rx}) [9]:

$$Z_{T_x} = R_{T_x} + j\omega L_{T_x} + \frac{1}{j\omega C_{T_x}}, \tag{4}$$

$$Z_{R_x} = R_L + R_{R_x} + j\omega L_{R_x} + \frac{1}{j\omega C_{R_x}}, \tag{5}$$

siendo R_L la resistencia de carga del receptor y R_{Rz} la resistencia del circuito resonante del receptor.

Las ecuaciones anteriores representan la impedancia de cada uno de los sistemas de bobinas, pero este sistema también lo vamos a modelar como un sistema de autoinducción, como el mostrado en Figura 1 (c). Para ello vamos a definir la impedancia de interacción entre ambos sistemas Z_{12} o Z_{21}, utilizando la inducción magnética entre dos bobinas (M), según la ecuación 6 [9]:

$$Z_{12} = Z_{21} = j\omega M. \tag{6}$$

Por otro lado, vamos a definir las impedancias resultantes de entrada y salida del sistema:

$$Z_{in} = Z_{T_x} - \frac{Z_{12}^2}{Z_{R_x}}, \tag{7}$$

$$Z_{out} = Z_{R_x} - \frac{Z_{12}^2}{Z_s + Z_{T_x}}. \tag{8}$$

Además de las anteriores ecuaciones de pueden calcular las corrientes y potencias de ambos sistemas, pudiendo obtener el coeficiente de eficiencia de transmisión de potencia. En este trabajo no vamos a profundizar tanto en esta área objeto de otros trabajos de investigación y aplicación.

Estas ecuaciones nos indican que el comportamiento del sistema conjunto puede reducirse a un circuito equivalente y que, si mantenemos un sistema fijo (el lector) las variaciones en R_{Rx}, L_{Rx} y C_{Rx} (la tarjeta o *tag*) van a alterar el funcionamiento global del sistema de forma medible.

4 NanoVNA

NanoVNA es un Analizador de Redes Vectorial (VNA) portátil y de bajo costo, creado inicialmente por edy555 y mejorado por otros colaboradores, siendo un proyecto libre [18]. Posteriormente, se ha creado la versión V2 que mejora el filtrado de las señales de entrada y amplía el rango de frecuencia, rediseñando el dispositivo desde cero, sin utilizar harmónicos para las mediciones y no utiliza el proyecto inicial de edy555 [19].

El dispositivo se compone de un medidor capaz de medir algunos parámetros S, como son S11 y S21, trae un juego de calibradores (abierto, corto, 50Ω) y permite mostrar las medidas en varias unidades, como onda estacionaria (SWR), valores de módulo y fase de cada canal, carta de Smith [18]. Además, incluye comunicación serie, mediante puerto USB, para conectarlo a un PC y procesar las medidas con software diseñado para tal función.

Para nuestro proyecto utilizaremos un dispositivo NanoVNA, donde utilizaremos el parámetro S11 para recoger los datos de reflexión obtenidos al realizar el acoplamiento entre el *tag* y la antena.

Figura 6. Dispositivo NanoVNA [20].

4.1 Parámetros S

Los parámetros S nos proporcionan una relación de entrada y salida entre puertos en un sistema eléctrico [21].

En sistemas de alta frecuencia es muy complicado realizar mediciones directas de tensión y corriente. Los parámetros S nos proporcionan una relación entre entrada y salida ofrecen un buen sistema para ver el comportamiento y realizar ajustes en los sistemas de radiofrecuencia.

Son utilizados en muchas aplicaciones de ingeniería como sistemas de comunicaciones, diseño de filtros, ajuste de antenas o diseño de PCBs.

En un sistema de dos puertos, los parámetros S se definen según la Figura 7.

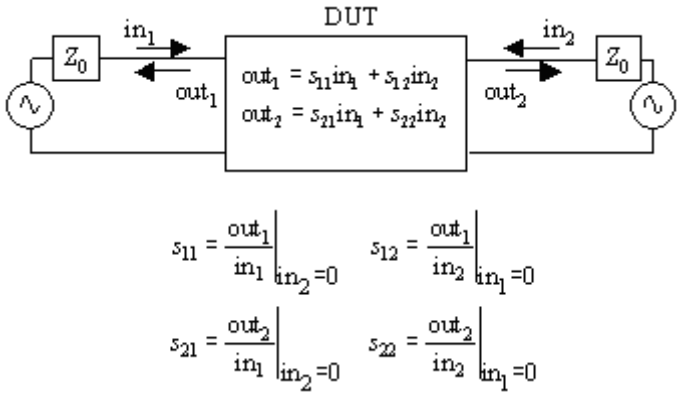

Figura 7. Sistema de dos puertos [21].

En la figura tenemos,

S_{11} el coeficiente de reflexión (tensión) en el puerto de entrada,

S_{12} la ganancia de tensión reflejada,

S_{21} la ganancia de tensión transmitida,

S_{22} el coeficiente de reflexión (tensión) en el puerto de salida.

Para el cálculo de la relación de onda estacionaria (SWR), se define según la siguiente ecuación:

$$SWR = \frac{1+|S_{11}|}{1-|S_{11}|}. \quad (9)$$

Para el objeto de nuestro proyecto, nos bastará utilizar S11 para el cálculo de la relación de onda estacionaria, que nos permitirá obtener el mínimo de reflexión cuando *tag* y antena, para una frecuencia dada, está en su mejor valor de acoplamiento, lo que implica obtener la "frecuencia de resonancia" del sistema.

5 Physically Unclonable Functions

La seguridad física, como la criptografía y autenticación, es un tema de interés que adquiere gran interés entre la comunidad de investigadores en materia de seguridad. Cada vez se investiga en conseguir modelos de seguridad que resulten inquebrantables ante usuarios malintencionados que mediante sucesivos ataques al sistema consiga saltarse los sistemas de defensa. El avance de la tecnología provee de nuevos sistemas y técnicas para romper las barreras criptográficas que dejan a los sistemas de autenticación y seguridad vulnerables ante cualquier usuario experimentado en un menor tiempo, comparado con unos años atrás.

Un sistema que cobra gran relevancia en seguridad es "*Physically Unclonable Functions* (PUF)". Las PUF añaden una serie de retos aleatorios impredecibles, o difíciles de predecir, de controlar o reproducir que aportan una capa de seguridad ante la autenticación de un usuario [22].

Las PUF están basadas en hardware, introduciendo variaciones físicas aleatorias en su construcción que resulten difíciles de predecir y reproducir. Algunas de estas variaciones físicas son debidas a las tolerancias de construcción, por ejemplo, de un circuito integrado, que hace que tenga estas características no clonables [23].

Cada PUF puede estar compuesto de un reto simple o de varios retos, utilizándose tuplas del tipo (Ci,Ri), siendo Ci el reto (*challenge*) y Ri la respuesta al reto i [23].

5.1 Ventajas y aplicaciones de las PUF

Las PUF nos proporcionan dos ventajas, seguridad y coste.

Proporciona un sistema de seguridad complejo y aleatorio asociado a claves digitales. Por ejemplo, un sistema de almacenamiento de claves que devuelve una respuesta única dependiente de la variación en el proceso de producción hardware [23].

Por otra parte, la segunda ventaja es el coste ya que este sistema permite "almacenar" claves directamente en el hardware sin necesidad de utilizar una memoria no volátil. Por ejemplo, podemos utilizar el clave *hardware* de una FPGA para cifrar/descifrar la información cargada en la FPGA, sin la necesidad de crear una memoria no volátil [23].

Una aplicación interesante para nuestro objeto de estudio es el uso de PUF con identificador almacenado en una memoria no volátil, es decir, nuestro sistema de *tags* NFC dispone de un identificador "único" almacenado en una memoria, por otra parte, el *hardware* tiene tolerancias de fabricación (circuito resonador, antena, acoplamiento inductivo…). Con estos dos elementos podemos validar mediante dos retos la autenticidad del usuario, el identificador almacenado en memoria y fácilmente quebrantable, con la frecuencia de resonancia del sistema, dependiente de la fabricación hardware.

6 Diseño experimental

El objetivo del proyecto es comprobar la funcionalidad y viabilidad de uso de la frecuencia de resonancia como método de identificación única de cada tarjeta. Con los resultados que se obtengan de las pruebas de laboratorio se podrá determinar la idoneidad de uso de este parámetro como método de identificación inclonable de usuarios.

Para el desarrollo de este proyecto se han tenido en cuenta y realizado un plan de medidas para comprobar diversos factores que pueden alterar el resultado final.

La primera fase de la investigación ha sido elegir los materiales a utilizar. Se ha escogido la plataforma nanoVNA por su facilidad en la adquisición y uso por cualquier tipo de usuario, así como su bajo costo. Este instrumento permite la adquisición y transferencia de datos a un PC. La antena elegida es específica para aplicaciones NFC, con geometría similar a los *tags* a utilizar, está fabricada por RFExplorer, empresa especializada en instrumentación de analizadores de espectro portátiles. Por último, el elemento con el que realizaremos el estudio, los *tags* NFC, en este caso no se ha optado por ninguna etiqueta en especial, haciendo más global el estudio para cualquier tipo de *tag* existente en el mercado.

En el apartado de desarrollo y pruebas se listará todos los elementos que se utilizan en el proyecto.

La siguiente fase del proyecto consiste en realizar un banco de pruebas, consistente en la toma de datos de catorce *tags*, en diferentes fechas y lugares para comprobar la variación en los datos obtenidos. Se realizan cinco rondas de comprobación de datos, repartidas en diferentes fechas entre abril y julio, y en diferentes zonas de España. La calibración previa se realiza en la zona de trabajo habitual, en el mes de abril, clima templado, humedad media, en zona de meseta y altitud de 850 metros sobre el nivel del mar. Tras la calibración se realiza la primera ronda de comprobación. La segunda ronda de comprobaciones está planificada su realización en el mes de mayo, en zona de costa del levante, a una altitud inferior de 100 metros, clima húmedo de costa y clima templado en el momento del ensayo. La tercera roda de comprobación se planifica para finales de junio en Madrid, con clima cálido, altitud sobre 650-700 metros y humedad baja, teniendo en cuenta que en la capital de España el espectro radioeléctrico está más saturado. La cuarta ronda de comprobación se planifica en un lugar de la costa cantábrica, en el mes de julio, con clima cálido, humedad alta y baja altitud. Por último, la quinta ronda se realiza de nuevo, a finales de julio, en la zona de trabajo habitual con clima cálido y humedad muy baja.

A pesar de la planificación de varios lugares, temperaturas, altitudes y climas, así como zonas con espectro más saturado, se cree que no son factores de gran relevancia que puedan influir en gran medida en los resultados del proyecto. El factor con mayor influencia en los resultados es la colocación milimétrica de la antena con el *tag* a

comprobar. Según pruebas previas es un factor que influye en gran medida en las medidas obtenidas, por esta causa, para minimizar la influencia de la colocación se realiza una raya de colocación en antena y *tag*, para colocar siempre en la "misma" posición el *tag* y la antena, aunque siempre puede haber una pequeña variación en la colocación que incluirá en la comprobación de cada *tag* y el resultado obtenido.

Como anticipo a los resultados que se obtienen en el estudio, en varios casos obtendremos resultados similares para la frecuencia de resonancia de cada *tag*, quizá la resolución del NanoVNA no sea suficiente para discriminar frecuencias de resonancia muy cercanas, necesitando instrumentación con mayor número de puntos de muestreo e incrementando el precio del proyecto. También pueden darse casos de coincidir las frecuencias de resonancia de varias tarjetas.

7 Desarrollo y pruebas

7.1 Materiales a utilizar

Para la realización de las pruebas de recogida de resultados se han empleado los siguientes elementos:

- NanoVNA
- Antena Circular *RF Explorer* de campo cercano
- 14 *Tags MiFare Classic* geometría circular
- PC Portátil Windows 10 (donde corre el programa desarrollado de evaluación)
- Lenguaje de programación C#

La elección de estos elementos viene dada por su necesidad en la mayoría de casos y su simplicidad en otros casos.

El uso de NanoVNA se utiliza para comprobar el acoplamiento entre sistema receptor y *tag* a evaluar [17], como se explicó en capítulos anteriores. La antena de RF Explorer ha sido seleccionada por ser una antena con la geometría idónea, concretamente el modelo RFEAH-25, a los *tags* circulares en objeto de estudio y su diseño está específicamente realizado para aplicaciones de campo cercano (NFC). Aunque su ancho de banda es muy amplio, es suficiente para nuestra aplicación [24].

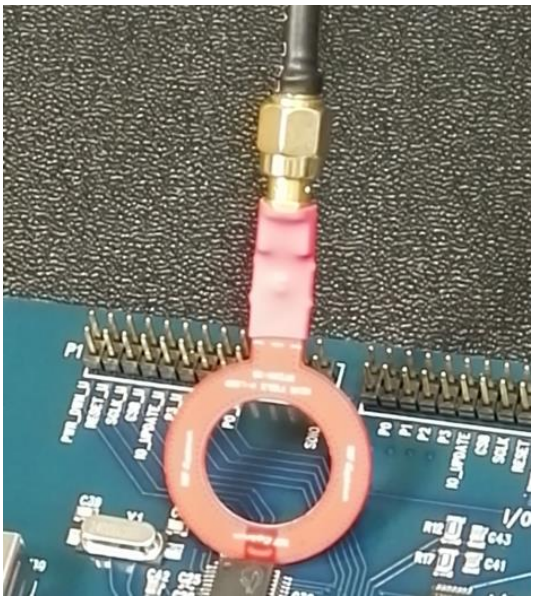

Figura 8. Antena RFEAH-25 [24].

Por otra parte, se ha utilizado un PC portátil con Windows 10 por su simplicidad y facilidad de desarrollo de software para este tipo de sistema operativo.

El lenguaje de programación elegido ha sido C# por ser un lenguaje de programación que domino con soltura y me facilita las labores de desarrollo, sin necesidad de aprender un nuevo lenguaje de programación. Microsoft dispone del IDE (Entorno de Desarrollo Integrado) Visual Studio, bastante potente que cuenta con amplio soporte para el lenguaje C#, con referencias a todas las funciones disponibles en versión web y, además, cuenta con una versión gratuita y libre de la comunidad [25].

7.2 Características de la aplicación de evaluación

Para el objeto de estudio la aplicación a desarrolla debe cumplir, como mínimo, las siguientes funciones:

- Cargar registros de calibración del NanoVNA para diferentes rangos del espectro de frecuencia.
- Consulta de frecuencias del rango de medición.
- Consulta de datos en bruto del rango de medición.
- Realizar una medida amplia y medidas más detalladas.
- Realizar una secuencia de medidas, una amplia y varias detalladas.
- Almacenar los datos obtenidos como una nueva tarjeta de referencia.
- Listar las tarjetas de referencia y el número de lecturas exitosas.
- Función de realizar un *check*/comparación de tarjeta objeto de estudio y tarjeta de referencia asociada, considerando un umbral de detección.

7.3 Desarrollo de la aplicación de evaluación

La aplicación de evaluación se ha realizado sobre una interfaz simple de consola de comandos. Aunque no sea visualmente bonita, la interfaz escogida es suficiente para navegar entre menús numéricos y mostrar datos de los experimentos realizados.

Tras crear un nuevo proyecto, se importan las bibliotecas necesarias para el uso del Puerto Serie y se realiza la primera pantalla a mostrar en la función *main* del programa, el menú de navegación numérico.

Se crea un objeto propio para el manejo, con mayor nivel de abstracción, del puerto serie, la clase *SerialPortComm*, el cual utiliza dentro de la clase creada la clase nativa *SerialPort* [26], del paquete System.IO.Ports. En esta clase se utiliza y define los métodos *Send* para enviar, *Receive* para recibir y la gestión de eventos producidos por la clase *SerialPort*. Esta *SerialPortComm* es invocada como un nuevo objeto desde el fichero principal del programa y se utilizará para recibir todos los datos necesarios para realizar los experimentos.

También se definen las clases *Card* y *CardTest*, que serán los objetos que modelen las tarjetas de referencia y tarjetas bajo estudio, la primera se compone de los siguientes atributos: ID para identificar la tarjeta, Descripción para añadir algún comentario sobre la misma, *CoarseIndex* que almacena el índice del valor con menor magnitud en la medida amplia, *CoarseSWRValue* guarda el valor de menor relación de onda estacionaria en la medida amplia, *CoarseFrequency* es el valor de frecuencia obtenida en la medida amplia, por otra parte están *FineIndex*, *FineSWRValue* y *FineFrequency* que son los mismos datos pero obtenidos en una medición detallada. La segunda clase se compone de los atributos *RealCardID* que es el identificador real de la tarjeta en estudio, *ReadCardID* es el identificador detectado por la aplicación en comparación con las tarjetas patrón, *ReadCard* almacena un objeto de la clase *Card* con los datos leídos, *IsMatchCard* almacena valores booleanos de coincidencia con el ID real e ID calculado y por último *ReadTime* almacena la marca temporal de lectura de la tarjeta bajo estudio.

Se incorpora la clase, *DataModel* que define el modelo de datos, listados de *Card* y *CardTest*, así como una clase auxiliar estática para el serializado y guardado de los datos en formato XML, el cual permite serializar/deserializar el modelo de datos para guardarlo o recuperarlo de un fichero.

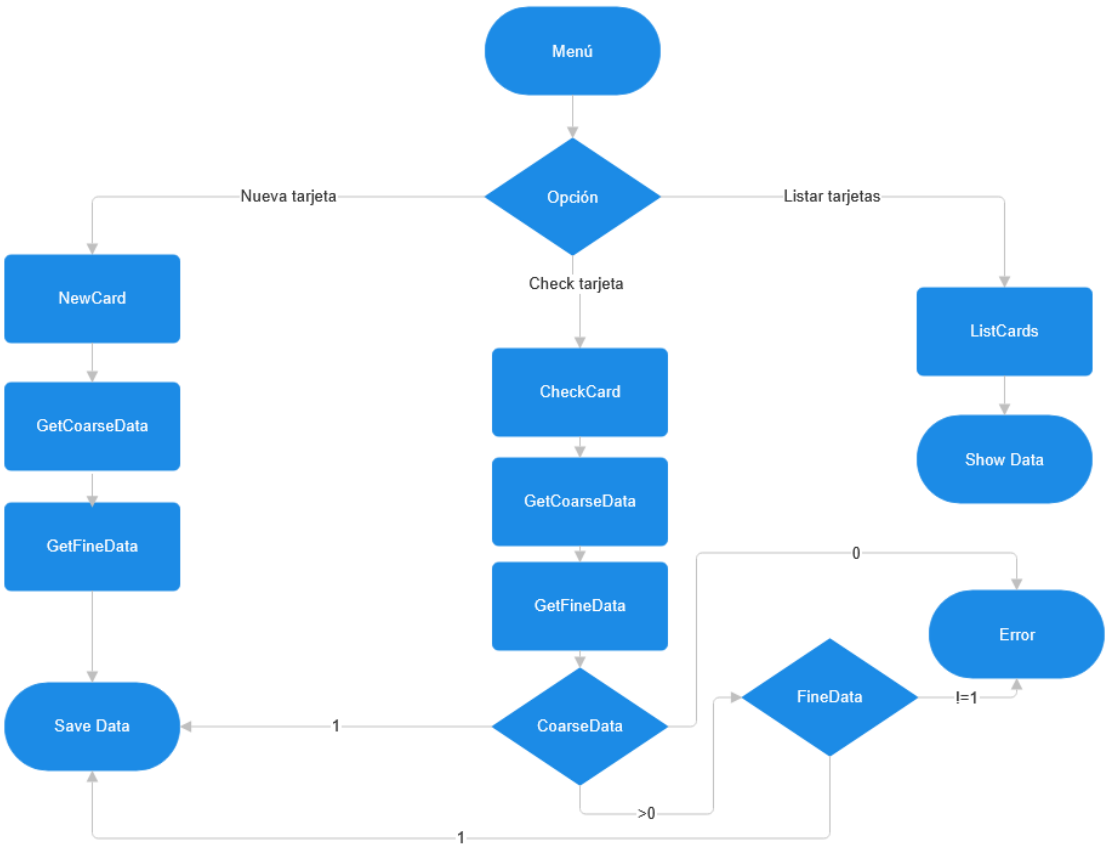

Figura 9. Diagrama de flujo simplificado de funcionamiento del programa.

7.3.1 Funciones desarrolladas

En los siguiente sub apartados se detallan aspectos de las funciones implementadas en el programa, en la Figura 9 se puede ver un diagrama de flujo simple que muestra la funcionalidad del programa utilizando las principales funciones.

7.3.1.1. GetFrequencies

Esta función se desarrolla como una función de comprobación para visualizar que desde la aplicación se puede obtener, sin ningún error ni problema, el listado de frecuencias asociados a cada uno de los puntos de análisis, dentro del rango de medición del NanoVNA. Para llevar a cabo esta operación se ha utilizado el comando "*frequencies*" [18] que se envía mediante comunicación serie y nos devuelve un listado el comando ejecutado más el listado, en cada línea, de cada una de las frecuencias del rango de medición, en total 101 frecuencias que es la resolución de puntos con la que trabaja NanoVNA [18].

7.3.1.2. GetData0

Esta función es similar a la anterior, se desarrolla para comprobar que se reciben los datos correctamente de las medidas S11. En este caso se utiliza el comando "*data 0*" y nos devuelve los valores leídos S11 del NanoVNA [18].

7.3.1.3. GetData1

Función completamente igual en funcionalidad a GetData0, con el matiz que en este caso obtenemos, mediante el comando "*data 1*" [18], los datos de S21 del NanoVNA.

7.3.1.4. Recall

Función desarrollada para cargar calibraciones guardadas en NanoVNA, admite un parámetro i, siendo el número de memoria a cargar. En este caso se envía el comando "*recall*" + i [18], y con ello el NanoVNA carga la configuración número i. Devuelve respuesta que podemos visualizarlo en pantalla.

7.3.1.5. FrequencyDataCmd

Esta función es llamada por otras funciones y sirve para enviar los comandos de obtención de listado de frecuencias y data 0 del NanoVNA. Envía, mediante comunicación serie RS232 los comandos "*frequency*" y "*data* 0" [18]. Una vez enviado

los comandos se realiza una espera de 500ms, con el objetivo de dar tiempo de procesamiento y respuesta al NanoVNA.

7.3.1.6. GetCoarseData

Esta función interna está desarrollada para ser llamada por las funciones de añadir una nueva tarjeta de referencia "*NewCard*" y la función de comprobación de tarjeta a detectar "*CheckCard*". Esta función carga la calibración de la memoria 0, que cubre todo el espectro posible, mediante la función "*Recall(0)*" y posteriormente llama a "*FrequencyDataCmd*" para obtener los datos de S11 del NanoVNA. Tras la llamada a estas dos funciones, se ejecuta un bucle de tipo "*while(true)*", que se ejecutará de manera infinita mientras haya datos disponibles que se reciban mediante RS232. Dentro del bucle llamamos a la función "*Read*" del objeto "*comPort*", el resultado lo almacenamos en una variable de tipo "*string*" que la utilizamos para comprobar el tipo de datos recibido, que se realiza posteriormente, dentro del bucle, mediante condicionales "*if*", determinando el proceso a realizar de lectura y procesado de datos, que se divide en el proceso 1, obtención y creación de una lista con los valores de frecuencias recibidas. El proceso 2, primero calcula el módulo resultante del número imaginario obtenido en los datos S11, y posteriormente se calcula el valor de relación de onda estacionaria (*SWR*), con la . (9, terminando con su almacenaje en una lista de valores de *SWR*, este procesamiento se realiza para cada uno de los valores correspondiente a cada frecuencia.

Una vez realizada la obtención y procesamiento de datos recibidos, se realiza un bucle para recorrer todos los valores de la lista de *SWR* con el objetivo de obtener el índice del menor valor de *SWR*, en el cual el sistema tendrá el mejor acoplamiento.

7.3.1.7. GetFineData

Esta es otra función interna similar a "*GetCoarseData*", es llamada por las mismas funciones y tiene un funcionamiento y procesamiento igual a la anterior función. Su objetivo es obtener datos con mayor resolución, obteniendo datos del mismo espectro, pero con un mayor número de puntos, es decir, empleamos cuatro segmentos de datos con 101 puntos cada uno de los segmentos.

La función realiza la siguiente secuencia de llamadas al inicio:

- *Recall(1)* → Carga los valores de calibración del primer segmento.
- *FrequencyDataCmd* → Solicita las frecuencias y datos S11 del segmento.
- *Recall(2)* → Carga los valores de calibración del segundo segmento.
- *FrequencyDataCmd* → Solicita las frecuencias y datos S11 del segmento.
- *Recall(3)* → Carga los valores de calibración del tercer segmento.

- *FrequencyDataCmd* → Solicita las frecuencias y datos S11 del segmento.
- *Recall(4)* → Carga los valores de calibración del cuarto segmento.
- *FrequencyDataCmd* → Solicita las frecuencias y datos S11 del segmento.

El resto de la función es exactamente igual a "*GetCoarseData*".

7.3.1.8. NewCard

Esta función nos permite añadir una nueva tarjeta de referencia al modelo de datos. Las tarjetas de referencia estarán identificadas por un ID y una descripción opcional y nos permitirán comparar las tarjetas a detectar con las tarjetas de referencia.

La función inicialmente crea un objeto del tipo de clase "*Card*", que posteriormente se irán configurando sus atributos. Los siguientes pasos que realiza la función es solicitar al usuario el ID de tarjeta y la descripción de la tarjeta. Terminada la solicitud de datos al usuario, se inicia la secuencia de obtención de datos con las funciones "*GetCoarseData*" y *GetFineData*". Seguirá con la eliminación de todas las tarjetas de referencia con el mismo ID a añadir, para evitar problemas posteriores, añade la tarjeta al listado de tarjetas de referencia y guarda el modelo de datos. También muestra los datos relevantes obtenidos de la tarjeta.

7.3.1.9. ListCards

Esta función permite al usuario mostrar los datos de las tarjetas. Muestra el listado de todas las tarjetas ordenada por ID, con la información del ID de tarjeta, número de veces comprobada, número de comprobaciones exitosas y número de comprobaciones erróneas.

Además, al terminar de mostrar las tarjetas, muestra la misma información con carácter global. Toda la información que proporciona esta función es con vista a las estadísticas y resultados obtenidos.

7.3.1.10. CheckCard

Esta función permite al usuario detectar la tarjeta en estudio. Solicita al usuario el ID de la tarjeta que estamos comprobando, para determinar si la detección es correcta o errónea, luego realizará un proceso de obtención y comparación de datos para determinar la tarjeta que está sobre la antena.

Genera un objeto del tipo de clase "*Card*" para almacenar los datos de frecuencias y *SWR*, y otro objeto del tipo clase "*CardTest*", que almacenará los datos de la detección y si fue exitosa o errónea la detección.

La función emplea las mismas funciones que se emplearon en "*NewCard*", obteniendo los mismos datos mediante "*GetCoarseData*" y "*GetFineData*". Tras obtener los datos de la tarjeta, comienza un proceso iterativo de detección de la tarjeta.

Inicialmente busca la tarjeta en los datos "*Coarse*", estableciendo un límite de 5 saltos por arriba y por debajo de 0,015MHz cada uno, es decir el umbral de limitación de tarjetas de referencia cercanas en este paso se limita a un total de ±0.075MHz, y posteriormente busca la tarjeta en los datos "Fine", mediante bucles que irán añadiendo una tolerancia de variación cada vez mayor hasta encontrar un dato coincidente, cada salto es de ±0,00375 MHz, hasta un umbral máximo de ±0,01875MHz.

Al finalizar la secuencia se muestra al usuario los datos de ID real de la tarjeta, el ID detectado y si son coincidentes. También puede darse el caso de haber varias coincidencias o que estén fuera de los umbrales de detección y no se podría determinar cuál es la tarjeta detectada.

8 Fase experimental y análisis

Para evaluar la viabilidad de puesta en marcha en un sistema real y la tasa de éxito, se ha realizado la evaluación de catorce tarjetas NFC, diferenciadas por un ID numérico de tarjeta para la fácil y correcta identificación, el UID y tipo de tarjeta para la elaboración de tabla de resultados y un banco de varias medidas, de las catorce tarjetas objeto de estudio, para comprobar la tasa de acierto del sistema.

El experimento se compone de varias fases:

1. Desarrollo de aplicación para evaluación de tarjetas.
2. Grabar los datos iniciales de las catorce tarjetas, denominadas tarjetas de referencia.
3. Realización de varias rondas completas de comprobación de las catorce tarjetas.
4. Listado de todas las tarjetas y evaluación de resultados.

El procedimiento para añadir una nueva tarjeta de referencia se realiza, primero poniendo la tarjeta a referenciar en el centro de la antena RFEAH-25 [24] tal y como podemos ver en la Figura 10. El siguiente paso es elegir la opción de "Nueva tarjeta" del menú de la aplicación, como podemos ver en la Figura 11 rellenamos los datos de ID y descripción de la tarjeta y esperamos a que realice el proceso de adquisición y procesado de datos, al finalizar se muestra los datos de la Figura 12.

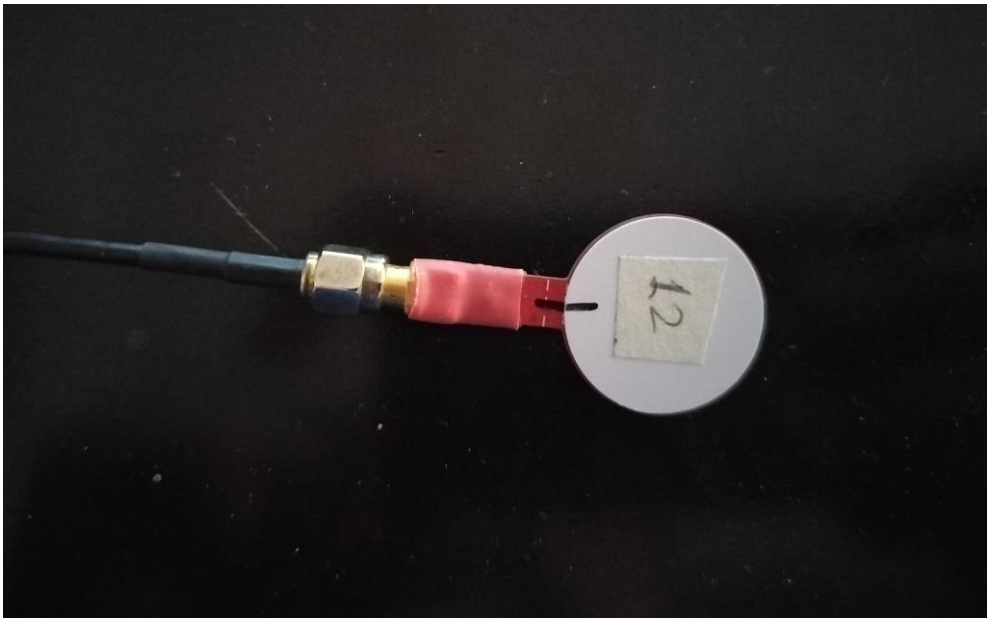

Figura 10. *Tag* NFC sobre antena RFEAH-25 para su análisis.

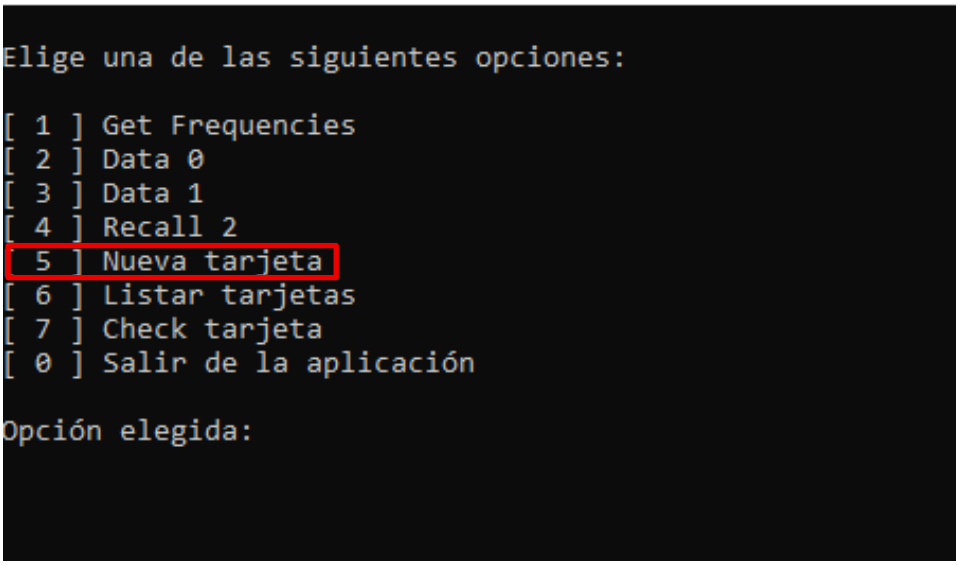

Figura 11. Menú de la aplicación con la opción "Nueva tarjeta".

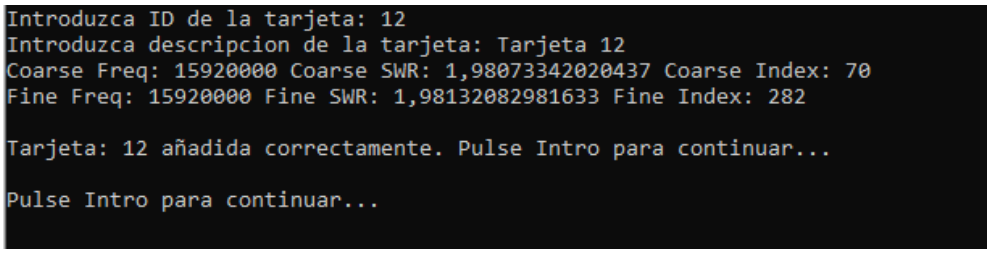

Figura 12. Resultado de añadir nueva tarjeta de referencia.

El procedimiento de detección de tarjetas es similar, se colocar la tarjeta a detectar en el centro de la antena RFEAH-25 [24] tal y como podemos observar en la Figura 10, el siguiente paso es elegir la opción de "*Check* tarjeta" del menú de la aplicación, como vemos en la Figura 13, tras elegir la opción, rellenamos el ID de la tarjeta que queremos detectar y esperamos a que se realice el proceso de adquisición, procesado de datos y detección de tarjeta, al finalizar nos mostrará los datos de detección de la tarjeta, de forma satisfactoria, según Figura 14, o de forma errónea, según Figura 15.

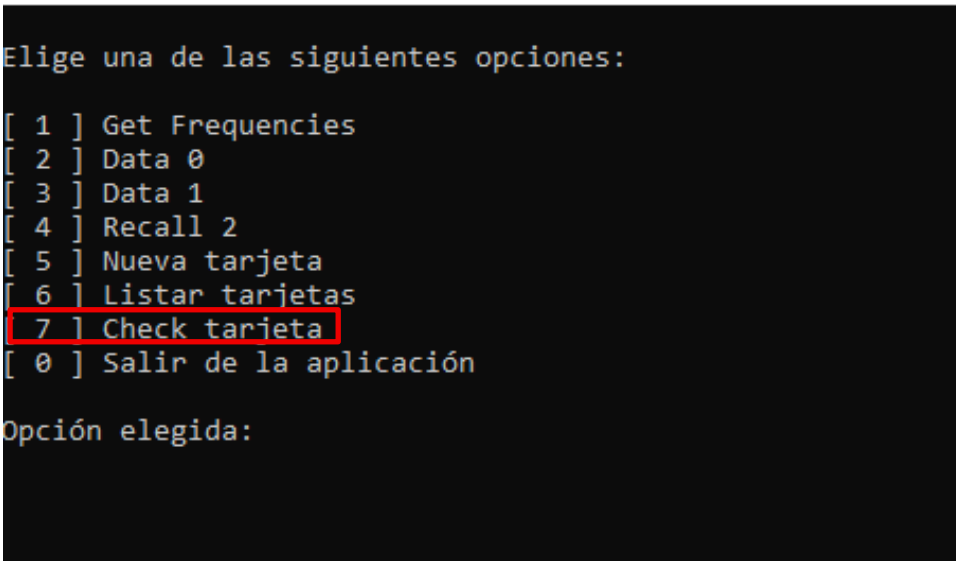

Figura 13. Menú de la aplicación con la opción “Check tarjeta”.

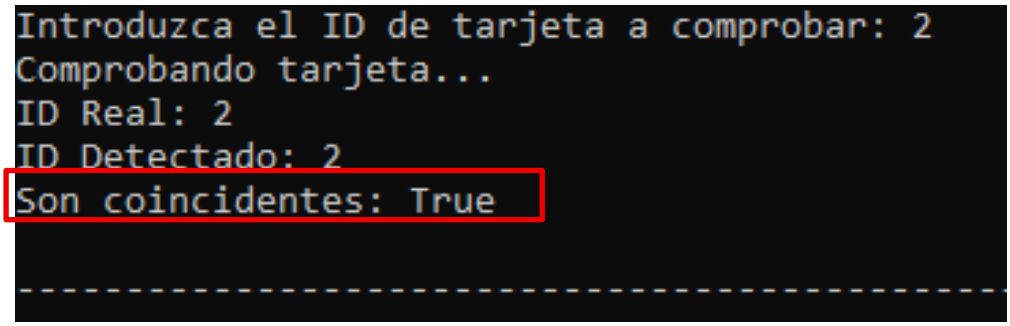

Figura 14. Resultado de detección “Éxito”.

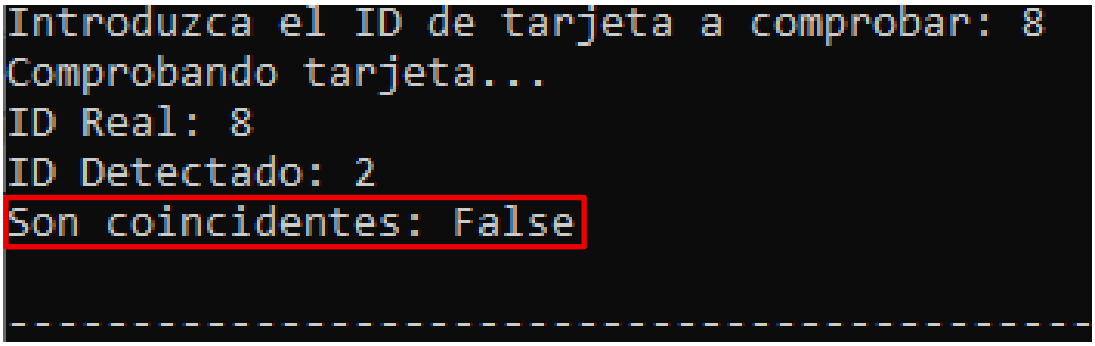

Figura 15. Resultado de detección “Error”.

El último procedimiento es el de listar todas las tarjetas en estudio y obtención de resultados, para ello en el menú elegiremos la opción “Listar tarjetas”, según la Figura 16. Tras acceder a la opción nos aparecerá en pantalla todos los datos de tarjetas según la Figura 17 y el resumen total de tarjetas comprobadas, operaciones exitosas y

operaciones erróneas tal y como observamos en la Figura 18. En la Tabla 1 se desglosan todas las tarjetas en estudio con sus datos y los valores obtenidos en el estudio para cada tarjeta. No obstante, en la valoración global de resultados, contando todos los ensayos como si fuera una única tarjeta, obtenemos un porcentaje de éxito del 50%, aunque individualizado el resultado es peor, siendo el éxito menor al 50%. A pesar de obtener en la mayoría de casos un porcentaje inferior al 50% de éxito, podemos considerar que cualquier valor superior al 0% en aciertos es ya de por sí un éxito del sistema, a nivel experimental, al poder haber detectado de forma satisfactoria la tarjeta en alguno de los ensayos. Comparándolo con la probabilidad aleatoria de detección, obtenemos que la probabilidad de éxito es uno entre catorce tarjetas, lo que supone una tasa de éxito de alrededor del 7,14 %. La eficiencia media en producción se podría mejorar haciendo un cribado previo de las tarjetas con un 0% de acierto. Esas se podrían volver a registrar o retirar. Se ha comprobado que en varios casos se han conseguido detectar tarjetas con una diferencia de frecuencias muy pequeña sin que el sistema las confundiera y, en la tarjetas que obtenemos un 0% de acierto creo que es debido a que se movieron durante el registro, generando el error obtenido en la detección, habría que comprobar si realizando un nueva registro el error disminuye, en otro caso esas tarjetas se pueden retirar.

A continuación, se realiza una lista de limitaciones del sistema y plan de análisis/ensayos de este sistema:

- Necesidad de colocación milimétricamente exacta de los *tags*.
- Limitación en puntos del nanoVNA que implica menor resolución de datos.
- No uso de varianzas previas en calibración (se utilizó, pero resultó ser un caos y se determinó utilizar una única muestra de patrón).
- Limitación de uso en sistema real como una opción de autenticación.
- Posibilidad de obtener frecuencias de resonancia muy cercanas (incapaz de distinción) o iguales para distintas tarjetas.
- Utilización de un solo instrumento de medida y antena para realizar el objeto de estudio del presente trabajo, sin tener en cuenta otra instrumentación o antena que pueda influir en los resultados.
- Limitación en el número de muestras realizadas, se han hecho pocas, pero suficientes para determinar el funcionamiento y viabilidad inicial del sistema.

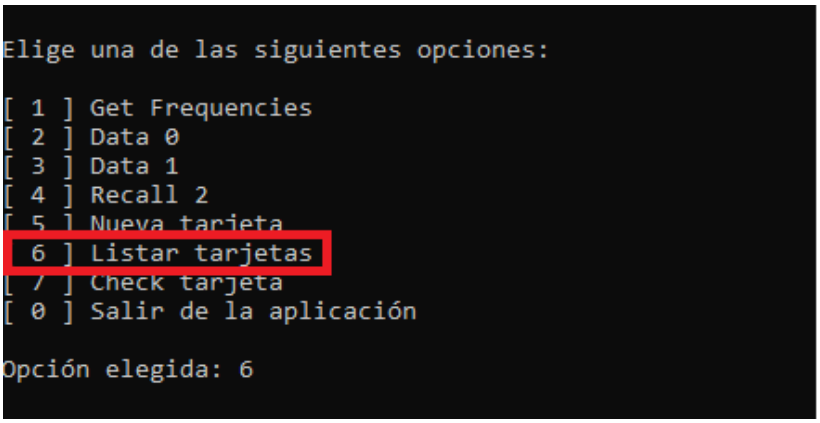

Figura 16. Menú de la aplicación con la opción “Listar tarjetas”.

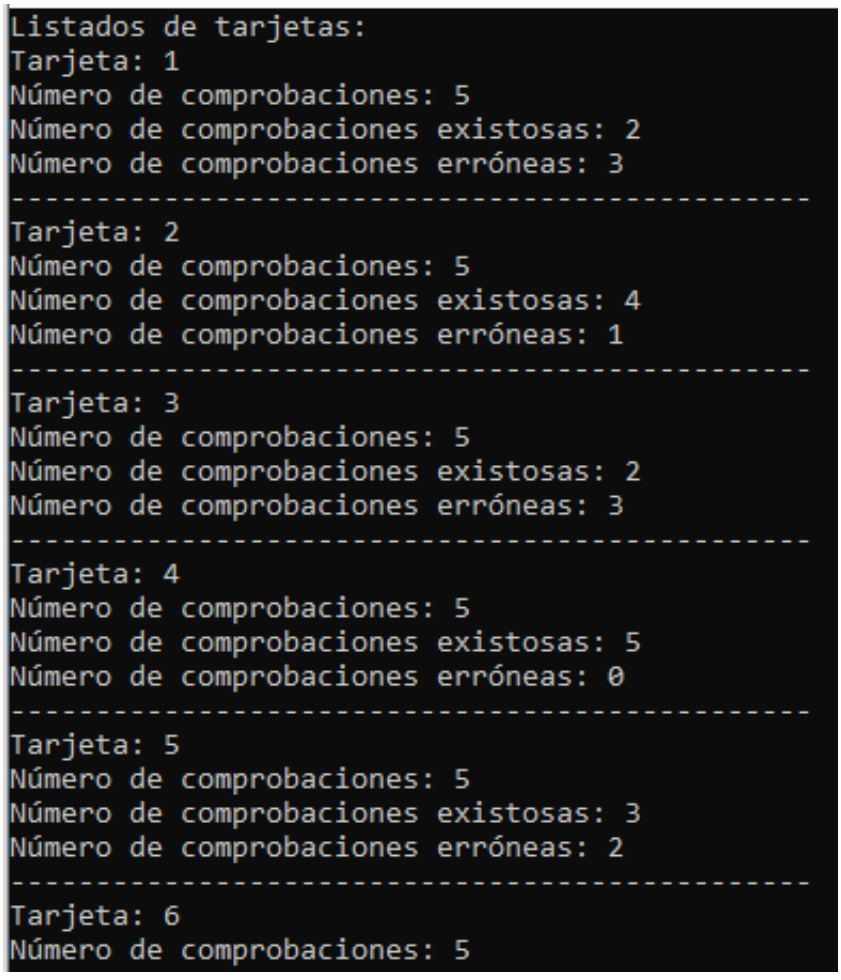

Figura 17. Listado de tarjetas comprobadas y tasa de éxito y error de cada tarjeta.

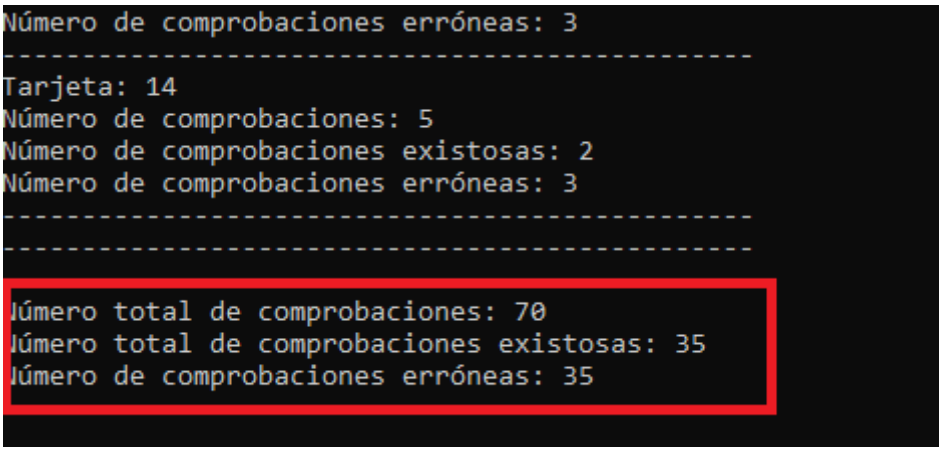

Figura 18. Resumen de comprobaciones totales, éxito y error.

ID Tarjeta	1	2	3	4	5	6	7
UID	047742B6E9 791F	047742339 2791F	04774289B9 791F	0477420626 E830	04774269D6 791F	0477423122 E830	0477429C36 7A1F
Frec. resonancia (MHz)	15.818750	15.995000	15.710000	15.188750	15.747500	15.443750	15.867500
Variación superior (MHz)	0.011250	0.011250	0	0.033750	0.015000	0.011250	0
Variación inferior (MHz)	0.015000	0.037500	0.018750	0.037500	0.022500	0.045000	0.030000
Número comprobaciones	5	5	5	5	5	5	5
Det.éxito	2	4	2	5	3	5	4
Det. error	3	1	3	0	2	0	1
% Éxito	40	80	40	100	60	100	80
% Error	60	20	60	0	40	0	20
ID Tarjeta	8	9	10	11	12	13	14
UID	0477421306 7A1F	047742BE6 D7A1F	0477426AB0 791F	04774216F47 91F	0477425CA D7A1F	04774252307 A1F	047742B0F0 791F
Frec. resonancia (MHz)	15.905000	15.691250	15.878750	15.946250	15.923750	15.687500	15.815000
Variación superior (MHz)	0.078750	0.003750	0.018750	0.015000	0	0.022500	0.060000
Variación inferior (MHz)	0.037500	0.022500	0.037500	0.026250	0.063750	0.007500	0.048750
Número comprobaciones	5	5	5	5	5	5	5
Det.éxito	0	1	2	3	0	2	2
Det. error	5	4	3	2	5	3	3
% Éxito	0	20	40	60	0	40	40
% Error	100	80	60	40	100	60	60

Tabla 1. Resumen de resultados.

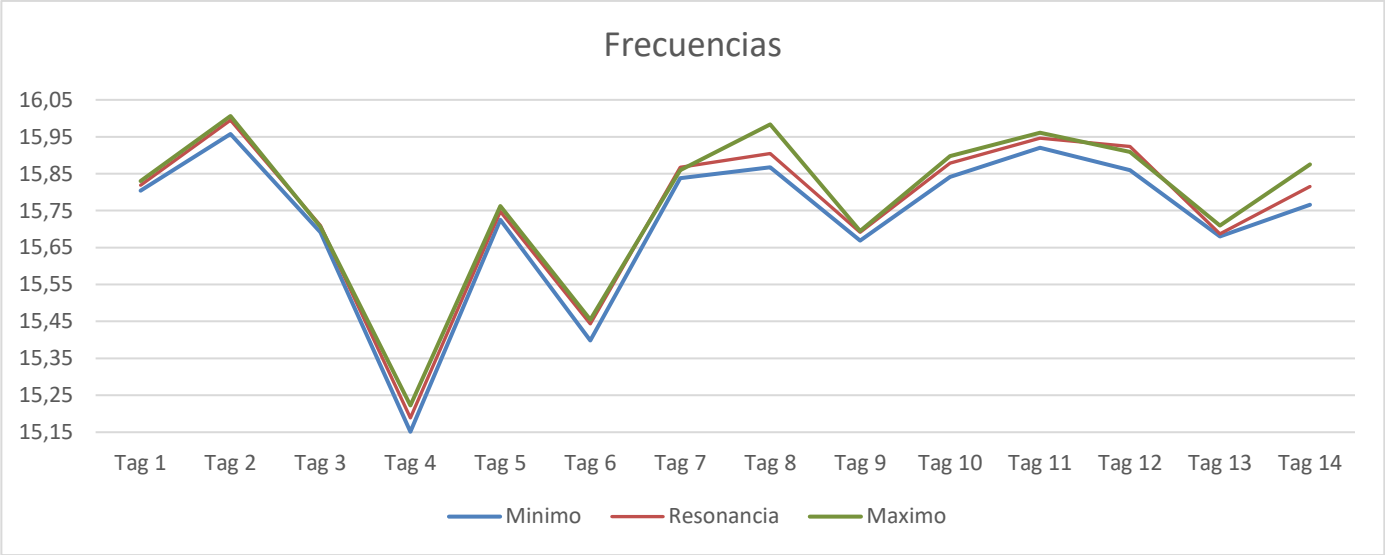

Gráfico 1. Distribución de frecuencia de resonancia, mínima y máxima de cada *tag.*

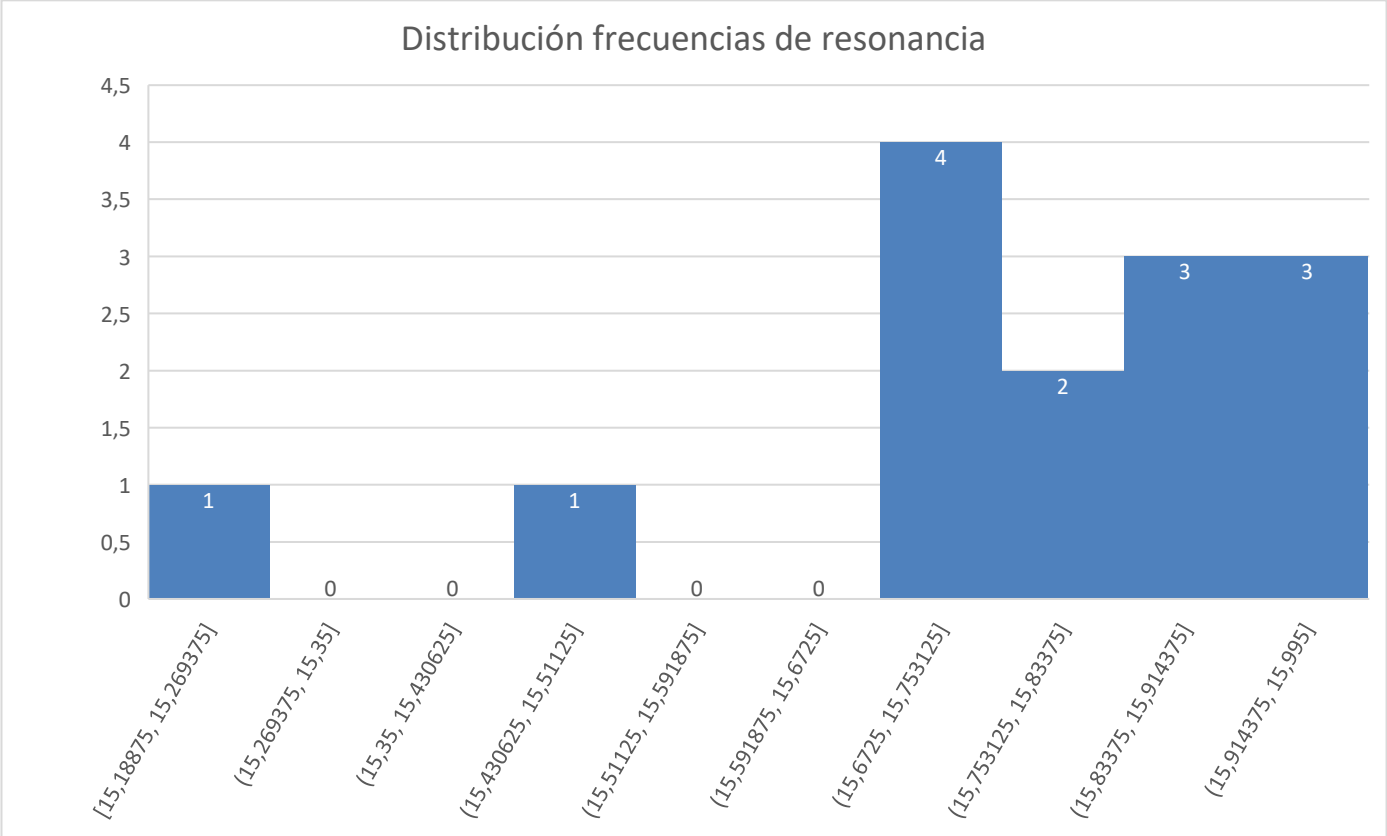

Gráfico 2. Histograma de distribución de las frecuencias de resonancia.

9 Otros ensayos

Durante la fase preliminar, se realizaron ensayos utilizando radio definida por software (SDR) y el instrumento hackRF, consistente en la transmisión de datos aleatorios hacía la tarjeta NFC y posterior recepción de los datos, analizando las variaciones producidas por el circuito resonador de la tarjeta. En esencia el estudio seguía una línea similar al estudio final, aprovechar las características de variación en procesos de construcción para obtener una huella digital única para cada tarjeta, que permita la autenticación inequívoca de los usuarios añadiendo una capa de seguridad no clonable al sistema NFC.

Se realizaron ensayos utilizados el dispositivo hackRF y el software libre GNURadio, con el cual se diseñó un sistema de transmisión/recepción definida por software y, con los datos adquiridos compartirlos con otra aplicación de análisis de datos que permitiera la detección de cada tarjeta.

Durante la fase experimental surgieron problemas con el sistema de ensayo realizado, la carga de procesamiento era tan grande que finalmente el sistema se quedaba congelado y resultó imposible llegar a transmitir y recibir en el mismo tiempo. Este hecho hizo que se tuviera que descartar este sistema de ensayo y precisó la búsqueda de un nuevo sistema experimental que permitiera realizar los ensayos de manera satisfactoria.

El sistema evolucionó al uso del elemento de análisis de bajo coste nanoVNA, basado en un esquema de transmisor/receptor definido por software y características de análisis vectorial con el cual se pueden analizar sistemas resonantes, comportamiento de antenas, filtros y otros elementos de radiofrecuencia.

10 Conclusiones

En el presente trabajo se ha investigado sobre el posible uso de la frecuencia de resonancia de un sistema NFC como huella digital, validando este parámetro como función física no clonable (PUF), con la cual dotar a los sistemas de autenticación NFC con un nivel mayor de seguridad al utilizar parámetros aleatorios de construcción [22].

Para obtener los datos de la investigación se ha realizado un programa básico en lenguaje C# de .Net para añadir tarjetas de referencia, comprobar la detección de esas tarjetas y obtener un listado de datos de las tarjetas objeto de estudio, como la frecuencia de resonancia del sistema, el máximo, el mínimo, porcentajes de éxito y de error.

Para líneas futuras de investigación, en el análisis no se ha tenido en cuenta diversos factores que pueden influir en el estudio como el uso de diferentes instrumentos de medida y captación de datos, uso de líneas de transmisión de diferentes longitudes o el uso de otro tipo de antenas. Tampoco se ha considerado utilizar como referencia, para cada tarjeta, un conjunto de medidas y análisis de varianza de la frecuencia de cada tarjeta previamente al proceso de detección.

Tras la obtención de los resultados, podemos concluir que, en el conjunto de las medidas, el porcentaje de éxito de detección es del cincuenta por ciento, dato muy útil y válido para verificar que es posible detectar tarjetas mediante la frecuencia de resonancia, pero no válido para ponerlo en marcha como método de autenticación en un sistema en producción. No obstante, el nanoVNA es un aparato de instrumentación y medida simple, de bajo coste y limitado en número de muestras (101 muestras), lo cual limita la precisión de los datos obtenidos [18]. Para aumentar la precisión se ha utilizado una calibración que cubre la totalidad del espectro NFC y cuatro subrangos que dividen el espectro en cuatro trozos iguales, que posteriormente se van uniendo para obtener una precisión mayor. Los resultados individualizados nos arrojan unas tasas de éxito peores que la valoración en su conjunto, añadiendo mayor tasa de error y dejando claro que el sistema no puede usarse en producción, pero para el objeto de estudio podemos considerar un éxito que exista un porcentaje de éxito superior a cero ya que en algún caso ha permitido detectar la tarjeta correctamente, de las catorce tarjetas utilizadas, únicamente dos han obtenido un cero por ciento de éxito. También hay que tener en cuenta que el sistema se ha analizado y validado en diferentes días y lugares, así como diferentes climas para ver si la variabilidad del sistema era suficientemente grande como para alterar la detección de los datos obtenidos.

El sistema tiene una serie de limitaciones expuestas en el capítulo de fase experimental y análisis que en líneas futuras de investigación deberían tenerse en cuenta para intentar eliminarlas, o en su caso, aminorarlas en la medida de lo posible. Otras de las líneas a seguir en el futuro es el análisis con grandes números de datos, aumentando las muestras de cada tarjeta, uso de varios valores de referencia para cada tarjeta y quizá

al análisis de la varianza tanto en la referencia como en la detección que permita definir con mayor exactitud la tarjeta detectada. También es interesante en un futuro, investigar con tarjetas de distinta geometría como puede ser la rectangular, sobre un soporte de inserción que coloque siempre y milimétricamente la tarjeta en la misma posición con respecto a la antena. El uso de un rango espectral más reducido para añadir mayor precisión o uso de otros analizadores de redes vectoriales más complejos puede ser útiles en próximos estudios. También considerar para el futuro el uso de tarjetas de tipo rectangular, con chip de memoria para utilizar un conjunto de mecanismos de validación doble, entre el uso de la huella digital y el almacenamiento de claves de autenticación del usuario.

Para finalizar, el estudio nos proporciona datos interesantes sobre la detección mediante la frecuencia de resonancia, capaz de distinguir diferentes tarjetas, pero no es viable su uso en producción, pero abre una línea de investigación útil para seguir investigando sobre este parámetro u otros parámetros no clonables que puedan ser interesantes para uso en un sistema de autenticación.

11 Referencias

[1] NFC Forum, «NFC & Other Wireless Technologies,» NFC Forum, [En línea]. Available: https://nfc-forum.org/learn/what-nfc-does. [Último acceso: 1 Junio 2024].

[2] M. V. Bueno Delgado, P. Pavón Mariño y A. de Gea García, «La tecnología NFC y sus aplicaciones en un entorno universitario,» Universidad Politécnica de Cartagena. Escuela Técnica Superior de Ingeniería de Telecomunicación, 2011.

[3] NFC Forum, «Use Cases,» NFC Forum, [En línea]. Available: https://nfc-forum.org/learn/use-cases/. [Último acceso: 3 Junio 2024].

[4] NFC Forum, «NFC Technology,» NFC Forum, [En línea]. Available: https://nfc-forum.org/learn/nfc-technology/. [Último acceso: 3 Junio 2024].

[5] M. Á. García Sanz, «Sistema NFC para el control de accesos,» Universidad de Valladolid - Escuela Técnica Superior de Ingeniería Informática, Valladolid, 2023.

[6] NXP Semiconductors, «MIFARE Plus SE - Secure contactless smart card IC for seamless migration,» 16 Marzo 2017. [En línea]. Available: https://www.nxp.com/docs/en/data-sheet/MF1SEP_H_10X1_SDS.pdf. [Último acceso: 1 Junio 2024].

[7] A. Taouirsa, «Hacking RFID, rompiendo la seguridad de Mifare (V),» Security Artwork, 10 Junio 2014. [En línea]. Available: https://www.securityartwork.es/2014/02/10/hacking-rfid-rompiendo-la-seguridad-de-mifare-v/. [Último acceso: 1 Junio 2024].

[8] Y. Gao, S. Al-Sarawi y D. Abbott, «Physical unclonable functions,» *Nature Electronics,* pp. 81-91, 2020.

[9] G. M. Plaizier, Design and modeling of a resonant inductively coupled Wireless Power Transfer system for micro aerial vehicles, The University of Utah, 2018.

[10] Keysight, «Resonance Frequency Formula: The Comprehensive Guide for Engineers,» [En línea]. Available: https://www.keysight.com/used/tw/en/knowledge/formulas/resonance-frequency-formula. [Último acceso: 02 08 2025].

[11] N. Forum, «Specifications,» NFC Forum, [En línea]. Available: https://nfc-forum.org/build/specifications. [Último acceso: 02 08 2025].

[12] R. Want, «Near field communication,» *IEEE Pervasive Computing,* vol. 10, nº 3, pp. 4-7, 2011.

[13] M. Barrios Barrigón, «Evaluación de la seguridad de tarjetas criptográficas,» Universidad de Valladolid - Escuela Técnica Superior de Ingenieros de Telecomunicación, Valladolid, 2018.

[14] International Organization for Standardization, *ISO/IEC 14443,* 2018.

[15] NFC Forum, «NFC Forum - What NFC does,» NFC Forum, [En línea]. Available: https://nfc-forum.org/learn/what-nfc-does/. [Último acceso: 22 junio 2024].

[16] N. Semiconductors, «MIFARE Classic EV1 1K - Mainstream contactless smart card IC for fast and easy solution development,» 23 Mayo 2018. [En línea]. Available: https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf. [Último acceso: 1 Junio 2024].

[17] P. Lathiya y J. Wang, «Near-Field Communications (NFC) for Wireless Power Transfer (WPT): An Overview,» de *Wireless Power Transfer - Recent Development Applications and New Perspectives*, IntechOpen, 2021.

[18] «NanoVNA,» [En línea]. Available: https://nanovna.com. [Último acceso: 12 12 2024].

[19] NanoRFE, «NanoVNA V2,» [En línea]. Available: https://nanorfe.com/nanovna-v2.html. [Último acceso: 12 12 2024].

[20] Q. Wang, W. Che, G. Monti y M. Mongiardo, «Measurements for Wireless Power Transfer by Using NanoVNA,» de *2021 XXXIVth General Assembly and Scientific Symposium of the International Union of Radio Science (URSI GASS)*, Roma, 2021.

[21] M. Garade, «EverythingRF,» 24 11 2018. [En línea]. Available: https://www.everythingrf.com/community/what-are-s-parameters. [Último acceso: 12 12 2024].

[22] S. Eiroa, I. Baturone, A. J. Acosta y J. Dávila, «Using physical unclonable functions for hardware authentication: a survey,» de *Proceedings XXV Conference on Design of Circuits and Integrated Systems (2010)*, Lanzarote, 2010.

[23] U. Rührmair y D. E. Holcomb, «PUFs at a glance,» de *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2014.

[24] R. Explorer, «RF Explorer Near Field Antenna Kit,» [En línea]. Available: https://j3.rf-explorer.com/rf-explorer-near-field-antenna-kit.html. [Último acceso: 10 06 2025].

[25] Microsoft, «Visual Studio: IDE y Editor de código para desarrolladores de software y Teams,» [En línea]. Available: https://visualstudio.microsoft.com/es/. [Último acceso: 10 06 2025].

[26] Microsoft, «SerialPort Clase (System.IO.Ports),» [En línea]. Available: https://learn.microsoft.com/es-es/dotnet/api/system.io.ports.serialport?view=netframework-4.8.1. [Último acceso: 10 06 2025].

12 Anexos

12.1 Anexo 1: Código fuente programa

12.1.1 Clase Menu (Principal)

```
public class Menu
    {
        private SerialPortComm comPort = new SerialPortComm();
        private DataModel dataModel = new DataModel();

        public Menu()
        {
            dataModel = DataModelFileHelper.Load<DataModel>();

            comPort.Open("COM5", "9600", "8", "One", "None", "None");

            while (true)
            {
                int userChoice;

                do
                {
                    Console.Clear();
                    Console.WriteLine("\nElige una de las siguientes opciones:\n");

                    Console.WriteLine("[ 1 ] Get Frequencies");
                    Console.WriteLine("[ 2 ] Data 0");
                    Console.WriteLine("[ 3 ] Data 1");
                    Console.WriteLine("[ 4 ] Recall 2");
                    Console.WriteLine("[ 5 ] Nueva tarjeta");
                    Console.WriteLine("[ 6 ] Listar tarjetas");
                    Console.WriteLine("[ 7 ] Check tarjeta");
                    Console.WriteLine("[ 0 ] Salir de la aplicación\n");

                    Console.Write("Opción elegida: ");

                } while (!int.TryParse(Console.ReadLine(), out userChoice) || userChoice
< 0 || userChoice > 7);
```

```
        Console.Clear();

        switch (userChoice)
        {
            case 1:
                GetFrequencies();
                break;
            case 2:
                GetData0();
                break;
            case 3:
                GetData1();
                break;
            case 4:
                Recall(2);
                break;
            case 5:
                NewCard();
                break;
            case 6:
                ListCards();
                break;
            case 7:
                CheckCard();
                break;
            case 0:
                Environment.Exit(0);
                break;
            default:
                Console.WriteLine("Try again!!");
                break;
        }
    }
}

public void GetFrequencies()
{
    comPort.Send(Encoding.Default.GetBytes("frequencies\r"));
```

```
    Thread.Sleep(100);

    List<string> str = new List<string>();

    while(true)
    {
        string data = comPort.Read();

        if(data == string.Empty)
        {
            break;
        }

        if (data.Contains("frequencies"))
        {
            Console.WriteLine("Frecuencias");
            continue;
        }

        str.Add(data);
        Console.WriteLine(data);

    }

    Console.ReadLine();
}

public void GetData0()
{
    comPort.Send(Encoding.Default.GetBytes("data 0\r\n"));
    Thread.Sleep(100);

    List<string> str = new List<string>();
    int counter = 0;

    while (counter < 101)
    {
        try
        {
```

```
                string data = comPort.Read();

                if (data.Contains("data 0"))
                {
                   Console.WriteLine("Data 0");
                   continue;
                }

                counter++;

                string[] values = data.Split(' ');
                double real = Convert.ToDouble(values[0],
CultureInfo.InvariantCulture);
                double imag = Convert.ToDouble(values[1],
CultureInfo.InvariantCulture);

                double module = Math.Sqrt(Math.Pow(real, 2) + Math.Pow(imag, 2));
                double swr = (1.0 + module) / (1.0 - module);

                str.Add(data);
                Console.Write("SWR: " + swr.ToString() + " ");
                Console.WriteLine(data);
             }
             catch(Exception e)
             {
                Console.WriteLine(e.Message);
             }
          }

          Console.ReadLine();
       }

       public void GetData1()
       {
          comPort.Send(Encoding.Default.GetBytes("data 1\r\n"));

          List<string> str = new List<string>();
          int counter = 0;
```

```
    while (counter < 101)
    {
        string data = comPort.Read();

        if (data.Contains("data 1"))
        {
            Console.WriteLine("Data 1");
            continue;
        }

        counter++;

        str.Add(data);
        Console.WriteLine(data);

    }

    Console.ReadLine();
}


public void Recall(int i)
{
    comPort.Send(Encoding.Default.GetBytes("recall " + i + "\r\n"));
    Thread.Sleep(500);
}

public void FrequencyDataCmd()
{
    comPort.Send(Encoding.Default.GetBytes("frequencies\r\ndata 0\r\n"));
    Thread.Sleep(500);
}

public bool GetCoarseData(Card card)
{
    int process = 0;
    List<int> freqs = new List<int>();
    List<double> swr = new List<double>();
    int minIndex = 0;
```

```
double minValue = 0;

try
{
   Recall(0);
   FrequencyDataCmd();

   while (true)
   {
      string data = comPort.Read();

      if (data == string.Empty)
      {
         break;
      }

      if (data.Contains("frequencies"))
      {
         process = 1;
         continue;
      }

      if (data.Contains("data 0"))
      {
         process = 2;
         continue;
      }

      if (process == 1)
      {
         int freq = Convert.ToInt32(data);
         freqs.Add(freq);
      }

      if (process == 2)
      {
         string[] values = data.Split(' ');
         double real = Convert.ToDouble(values[0],
CultureInfo.InvariantCulture);
```

```
                    double imag = Convert.ToDouble(values[1],
CultureInfo.InvariantCulture);
                    double module = Math.Sqrt(Math.Pow(real, 2) + Math.Pow(imag,
2));
                    double swr_value = (1.0 + module) / (1.0 - module);
                    swr.Add(swr_value);
                }
            }

            minIndex = 0;
            minValue = swr[0];

            for (int j = 0; j < swr.Count; j++)
            {
                if (swr[j] < minValue)
                {
                    minValue = swr[j];
                    minIndex = j;
                }
                else if (swr[j] == minValue && j != 0)
                {
                    j = j;
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine("Error en GetCoarseData");
            return false;
        }

        card.CoarseFrequency = freqs[minIndex];
        card.CoarseIndex = minIndex;
        card.CoarseSWRValue = minValue;

        return true;
    }

    public bool GetFineData(Card card)
```

```
{
   int process = 0;
   List<int> freqs = new List<int>();
   List<double> swr = new List<double>();
   int minIndex = 0;
   double minValue = 0;

   try
   {
      // Sub partición 1
      Recall(1);
      FrequencyDataCmd();

      // Sub partición 2
      Recall(2);
      FrequencyDataCmd();

      // Sub partición 3
      Recall(3);
      FrequencyDataCmd();

      // Sub partición 4
      Recall(4);
      FrequencyDataCmd();

      while (true)
      {
         string data = comPort.Read();

         if (data == string.Empty)
         {
            break;
         }
         if (data.Contains("recall"))
         {
            continue;
         }

         if (data.Contains("frequencies"))
```

```
                {
                    process = 1;
                    continue;
                }

                if (data.Contains("data 0"))
                {
                    process = 2;
                    continue;
                }

                if (process == 1)
                {
                    int freq = Convert.ToInt32(data);
                    freqs.Add(freq);
                }

                if (process == 2)
                {
                    string[] values = data.Split(' ');
                    double real = Convert.ToDouble(values[0],
CultureInfo.InvariantCulture);
                    double imag = Convert.ToDouble(values[1],
CultureInfo.InvariantCulture);
                    double module = Math.Sqrt(Math.Pow(real, 2) + Math.Pow(imag,
2));
                    double swr_value = (1.0 + module) / (1.0 - module);
                    swr.Add(swr_value);
                }
            }

            minIndex = 0;
            minValue = swr[0];

            for (int j = 0; j < swr.Count; j++)
            {
                if (swr[j] < minValue)
                {
                    minValue = swr[j];
```

```
                minIndex = j;
            }
            else if (swr[j] == minValue && j != 0)
            {
                j = j;
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("Error en GetFineData");
        return false;
    }

    card.FineFrequency = freqs[minIndex];
    card.FineIndex = minIndex;
    card.FineSWRValue = minValue;
    return true;
}

public void NewCard()
{
    Card card = new Card();

    Console.Write("Introduzca ID de la tarjeta: ");
    int id;
    if(int.TryParse(Console.ReadLine(), out id))
    {
        card.ID = id;
    }

    Console.Write("Introduzca descripcion de la tarjeta: ");
    string descripcion = Console.ReadLine();
    card.Description = descripcion;

    if(!GetCoarseData(card) || !GetFineData(card))
    {
        Console.ReadLine();
        return;
```

```
            }

            dataModel.Cards.RemoveAll(x => x.ID == card.ID);
            dataModel.Cards.Add(card);
            DataModelFileHelper.Save(dataModel);

            Console.WriteLine("Coarse Freq: " + card.CoarseFrequency +" Coarse SWR: " + card.CoarseSWRValue +" Coarse Index: " + card.CoarseIndex);
            Console.WriteLine("Fine Freq: " + card.FineFrequency + " Fine SWR: " + card.FineSWRValue + " Fine Index: " + card.FineIndex);
            Console.WriteLine("");
            Console.WriteLine("Tarjeta: " + card.ID + " añadida correctamente. Pulse Intro para continuar...");
            Console.WriteLine("");
            Console.WriteLine("Pulse Intro para continuar...");

            Console.ReadLine();
        }

        public void ListCards()
        {
            Console.WriteLine("Listados de tarjetas: ");

            foreach(Card card in dataModel.Cards.OrderBy(r => r.ID))
            {
                List<CardTest> cards = (from r in dataModel.TestCards where r.RealCardID == card.ID select r).ToList();

                Console.Write("Tarjeta: ");
                Console.WriteLine(card.ID);
                Console.Write("Frecuencia resonancia: ");
                Console.WriteLine(card.FineFrequency);
                Console.Write("Número de comprobaciones: ");
                Console.WriteLine(cards.Count);
                Console.Write("Número de comprobaciones existosas: ");
                Console.WriteLine(cards.Where(r => r.ReadCardID == card.ID).Count());
                Console.Write("Número de comprobaciones erróneas: ");
                Console.WriteLine(cards.Where(r => r.ReadCardID != card.ID).Count());
```

```
                Console.Write("Frecuencia mínima: ");
                Console.WriteLine(cards.OrderBy(x =>
x.ReadCard.FineFrequency).First().ReadCard.FineFrequency);
                Console.Write("Frecuencia máxima: ");
                Console.WriteLine(cards.OrderBy(x =>
x.ReadCard.FineFrequency).Last().ReadCard.FineFrequency);
                Console.Write("Variación por debajo: ");
                Console.WriteLine(card.FineFrequency - cards.OrderBy(x =>
x.ReadCard.FineFrequency).First().ReadCard.FineFrequency);
                Console.Write("Variación por encima: ");
                Console.WriteLine(cards.OrderBy(x =>
x.ReadCard.FineFrequency).Last().ReadCard.FineFrequency - card.FineFrequency);
                Console.WriteLine("-----------------------------------------");
            }

            Console.WriteLine("-----------------------------------------");
            Console.WriteLine("-----------------------------------------");
            Console.Write("Número total de comprobaciones: ");
            Console.WriteLine(dataModel.TestCards.Count);
            Console.Write("Número total de comprobaciones existosas: ");
            Console.WriteLine(dataModel.TestCards.Where(r => r.IsCardMatch ==
true).Count());
            Console.Write("Número de comprobaciones erróneas: ");
            Console.WriteLine(dataModel.TestCards.Where(r => r.IsCardMatch ==
false).Count());
            Console.WriteLine("-----------------------------------------");
            Console.ReadLine();
        }

        public void CheckCard()
        {
            Card card = new Card();
            CardTest cardTest = new CardTest();

            cardTest.ReadTime = DateTime.Now;

            Console.Write("Introduzca el ID de tarjeta a comprobar: ");
            int id_real;
            if (int.TryParse(Console.ReadLine(), out id_real))
```

```
            {
                cardTest.RealCardID = id_real;
            }

            Console.WriteLine("Comprobando tarjeta... ");

            if (!GetCoarseData(card) || !GetFineData(card))
            {
                Console.ReadLine();
                return;
            }

            cardTest.ReadCard = card;

            List<Card> cardCoarseMatch = null;
            List<Card> cardFineMatch = null;

            for(int i = 5; i >= 0; i--)
            {
                List<Card> cards = (from x in dataModel.Cards where x.CoarseIndex <=
card.CoarseIndex + i && x.CoarseIndex >= card.CoarseIndex - i select x).ToList();

                if(cards.Count > 0)
                {
                    cardCoarseMatch = cards;
                }

                if(cardCoarseMatch != null && cardCoarseMatch.Count == 1)
                {
                    break;
                }
            }

            if(cardCoarseMatch != null && cardCoarseMatch.Count == 1)
            {
                cardTest.ReadedCardID = cardCoarseMatch.First<Card>().ID;
            }
            else if(cardCoarseMatch != null && cardCoarseMatch.Count == 0)
            {
```

```
                Console.WriteLine("Ninguna tarjeta encontrada... Pulse intro para
continuar...");
                Console.ReadLine();
                return;
            }
            else
            {
                if(cardCoarseMatch != null)
                {
                    for (int i = 5; i >= 0; i--)
                    {
                        List<Card> cards = (from x in cardCoarseMatch where x.FineIndex
<= card.FineIndex + i && x.FineIndex >= card.FineIndex - i select x).ToList();

                        if (cards.Count > 0)
                        {
                            cardFineMatch = cards;
                        }

                        if (cardFineMatch != null && cardFineMatch.Count == 1)
                        {
                            break;
                        }
                    }
                }

                if (cardFineMatch != null && cardFineMatch.Count == 1)
                {
                    cardTest.ReadCardID = cardFineMatch.First<Card>().ID;
                }
                else if (cardFineMatch != null && cardFineMatch.Count == 0)
                {
                    Console.WriteLine("Ninguna tarjeta encontrada... Pulse intro para
continuar...");
                    Console.ReadLine();
                    return;
                }
                else
                {
```

```
                    Console.WriteLine("Se encontraron varias coincidencias, no es posible
determinar la tarjeta. Pulse intro para continuar...");
                    Console.ReadLine();
                    return;
                }
            }

            if(cardTest.ReadCardID == cardTest.RealCardID)
            {
                cardTest.IsCardMatch = true;
            }
            else
            {
                cardTest.IsCardMatch = false;
            }

            dataModel.TestCards.Add(cardTest);
            DataModelFileHelper.Save(dataModel);

            Console.Write("ID Real: ");
            Console.WriteLine(cardTest.RealCardID);
            Console.Write("ID Detectado: ");
            Console.WriteLine(cardTest.ReadCardID);
            Console.Write("Son coincidentes: ");
            Console.WriteLine(cardTest.IsCardMatch.ToString());
            Console.WriteLine();
            Console.WriteLine("----------------------------------------------------------------
----------------------");
            Console.WriteLine();
            Console.ReadLine();
        }

    }
```

12.1.2 Clase SerialPortComm (Manejo del Puerto Serie)

```
public class SerialPortComm
  {
    public SerialPort sp = new SerialPort();

    private object thisLock = new object();

    public event SerialPortEventHandler comReceiveDataEvent;

    public event SerialPortEventHandler comOpenEvent;

    public event SerialPortEventHandler comCloseEvent;


    public void Open(string portName, string baudRate, string dataBits, string stopBits, string parity, string handshake)
    {
                if (sp.IsOpen)
                {
                        Close();
                }

                sp.PortName = portName;
                sp.BaudRate = Convert.ToInt32(baudRate);
                sp.DataBits = Convert.ToInt16(dataBits);

                if (handshake == "None")
                {
                        sp.RtsEnable = true;
                        sp.DtrEnable = true;
                }

                SerialPortEventArgs serialPortEventArgs = new SerialPortEventArgs();

                try
                {
                        sp.StopBits = (StopBits)Enum.Parse(typeof(StopBits), stopBits);
```

```
                    sp.Parity = (Parity)Enum.Parse(typeof(Parity), parity);
                    sp.Handshake = (Handshake)Enum.Parse(typeof(Handshake), handshake);
                    sp.WriteTimeout = 1000;
                    sp.ReadTimeout = 1000;
                    sp.Open();
    //              sp.DataReceived += DataReceived;
                    serialPortEventArgs.isOpen = true;
                }
                catch (Exception)
                {
                    serialPortEventArgs.isOpen = false;
                }

                comOpenEvent?.Invoke(this, serialPortEventArgs);
            }

            public void Close()
            {
                new Thread(CloseSpThread).Start();
            }

            private void CloseSpThread()
            {
                SerialPortEventArgs serialPortEventArgs = new SerialPortEventArgs();
                serialPortEventArgs.isOpen = false;

                try
                {
                    sp.Close();
                    sp.DataReceived -= DataReceived;
                }
                catch (Exception)
                {
                    serialPortEventArgs.isOpen = true;
                }
```

```
                comCloseEvent?.Invoke(this, serialPortEventArgs);
        }

        private void DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {
                if (sp.BytesToRead > 0)
                {
                        lock (thisLock)
                        {
                                int bytesToRead = sp.BytesToRead;
                                byte[] array = new byte[bytesToRead];

                                try
                                {
                                        sp.Read(array, 0, bytesToRead);
                                }
                                catch (Exception)
                                {
                                }

                                SerialPortEventArgs  serialPortEventArgs  =
new SerialPortEventArgs();
                                serialPortEventArgs.receivedBytes = array;

                                comReceiveDataEvent?.Invoke(this,
serialPortEventArgs);
                        }
                }
        }

        public bool Send(byte[] bytes)
        {
                if (!sp.IsOpen)
                {
                        return false;
                }

                try
```

```
                {
                        sp.Write(bytes, 0, bytes.Length);
                }
                catch (Exception)
                {
                        return false;
                }

                return true;
        }

        public string Read()
    {
                if(sp.BytesToRead > 0)
       {
                        try
                        {
                                string received_data = sp.ReadLine();
                                return          received_data.Replace("ch>",
"").Trim();
                        }
                        catch (Exception)
                        {
                                return string.Empty;
                        }
                }
                else
       {
                        return string.Empty;
       }

    }

   }

public delegate void SerialPortEventHandler(object sender, SerialPortEventArgs e);
```

12.1.3 Clase SerialPortEventArgs (Eventos del Puerto Serie)

```
public class SerialPortEventArgs : EventArgs
   {
      public bool isOpen;
      public byte[] receivedBytes;
   }
```

12.1.4 Clase Card (Almacena los datos de cada tarjeta)

```
public class Card
   {
      public int ID { get; set; }
      public string Description { get; set; }
      public int CoarseFrequency { get; set; }
      public int CoarseIndex { get; set; }
      public double CoarseSWRValue { get; set; }
      public int FineFrequency { get; set; }
      public int FineIndex { get; set; }
      public double FineSWRValue { get; set; }
   }
```

12.1.5 Clase CardTest (Datos de la tarjeta detectada)

```
public class CardTest
   {
      public int RealCardID { get; set; }
      public int ReadCardID { get; set; }
      public Card ReadCard { get; set; }
      public bool IsCardMatch { get; set; } = false;
      public DateTime ReadTime { get; set; } = DateTime.Now;
   }
```

12.1.6 Clase DataModel (Modelo de datos)

```
public class DataModel
   {
      public List<Card> Cards = new List<Card>();
      public List<CardTest> TestCards = new List<CardTest>();
   }
```

12.1.7 Clase DataModelFileHelper (Guardado y recuperacíon del modelo de datos)

```
public static class DataModelFileHelper
   {
      public static string CurrentFile = AppDomain.CurrentDomain.BaseDirectory +
"Card.data";

      /// <summary>
      /// Serializes an object.
      /// </summary>
      /// <typeparam name="T"></typeparam>
      /// <param name="serializableObject"></param>
      /// <param name="fileName"></param>
      public static void SerializeObject<T>(T serializableObject, string fileName)
      {
         if (serializableObject == null) { return; }

         try
         {
            XmlDocument xmlDocument = new XmlDocument();
            XmlSerializer serializer = new
XmlSerializer(serializableObject.GetType());
            using (MemoryStream stream = new MemoryStream())
            {
               serializer.Serialize(stream, serializableObject);
               stream.Position = 0;
               xmlDocument.Load(stream);
               xmlDocument.Save(fileName);
            }
         }
```

```
    catch (Exception)
    {
        //Log exception here
    }
}


/// <summary>
/// Deserializes an xml file into an object list
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="fileName"></param>
/// <returns></returns>
public static T DeSerializeObject<T>(string fileName)
{
    if (string.IsNullOrEmpty(fileName)) { return default(T); }

    T objectOut = default(T);

    try
    {
        XmlDocument xmlDocument = new XmlDocument();
        xmlDocument.Load(fileName);
        string xmlString = xmlDocument.OuterXml;

        using (StringReader read = new StringReader(xmlString))
        {
            Type outType = typeof(T);

            XmlSerializer serializer = new XmlSerializer(outType);
            using (XmlReader reader = new XmlTextReader(read))
            {
                objectOut = (T)serializer.Deserialize(reader);
            }
        }
    }
    catch (Exception)
    {   }
```

```
        return objectOut;
    }

    public static void Save<T>(T dataModel)
    {
        if (CurrentFile != null && CurrentFile != "")
        {
            // Guardamos el modelo de datos serializado en el fichero.
            SerializeObject<T>(dataModel, CurrentFile);
        }
    }

    public static T Load<T>()
    {
        if (CurrentFile != null && CurrentFile != "")
        {
            // Guardamos el modelo de datos serializado en el fichero.
            return DeSerializeObject<T>(CurrentFile);
        }
        return default(T);
    }
}
```