ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Protocolo de comunicación sobre tecnologías sin contacto NFC basado en NDEF para plataformas web móviles

Communication protocol on contactless technologies NFC based on NDEF for mobile web platforms

Para acceder al Titulo de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Adrián Fernández Regaliza

Septiembre - 2024

Agradecimientos

Quiero expresar mi más profundo agradecimiento a todas aquellas personas que me han acompañado y apoyado a lo largo de este camino. En primer lugar, a mis padres, por su amor incondicional, su apoyo constante y por enseñarme el valor del esfuerzo y la perseverancia. A mi familia, por estar siempre presente, brindándome su confianza y ánimos en cada momento difícil. A mis amigos, por compartir este recorrido y hacer más llevadero el día a día con su compañía y buen humor. Y a mi pareja, por su paciencia, comprensión y por ser una fuente inagotable de motivación.

Finalmente, quisiera agradecer a mi tutor Jorge, por su orientación, profesionalidad y paciencia a lo largo de este proyecto, facilitando el proceso con sus consejos prácticos y efectivos.

Resumen

En la actual era digital, el empleo de dispositivos móviles para la interacción con el entorno cercano del usuario es cada vez más patente. La transferencia segura de información se lleva a cabo empleando tecnologías como Bluetooth o NFC (Near Field Communication). Es esta última la tecnología la que, en combinación con las tarjetas inteligentes, ha hecho despegar soluciones como los págos móviles, la firma electrónica o la autenticación robusta.

Sin embargo, estas soluciones son únicamente viables mediante el desarrollo de aplicaciones específicas para cada ecosistema móvil. A pesar de que la implantación soluciones basadas en plataformas web móviles están teniendo un rápido y gran crecimiento, actualmente no se dispone de una metodología universal e interoperable de comunicación que permita realizar un intercambio de datos entre una aplicación web móvil y una aplicación residente en una tarjeta inteligente.

Si bien, las API hacen posible leer o escribir datos en la memoria de las tarjetas en un formato específico (NDEF), no es viable la interacción abierta, la ejecución de comandos nativos, etc. sobre ella.

El presente trabajo plantea el desarrollo de un protocolo de comunicación y una aplicación para tarjeta inteligente, que haciendo uso de las API existentes, que elimine las restricciones y habilite la comunicación estándar y de bajo nivel con la tarjeta y pueda hacer uso de todas sus capacidades de cómputo.

Palabras clave: Tarjeta inteligente, NDEF, NFC

Abstract

In the current digital age, the use of mobile devices for interaction with the user's immediate environment is increasingly evident. The secure transfer of information is carried out using technologies such as Bluetooth or NFC (Near Field Communication). Is this latest technology which, in combination with cards smart, has made solutions such as mobile payments take off, electronic signature or strong authentication.

However, these solutions are only viable through development of specific applications for each mobile ecosystem. TO Although the implementation of solutions based on web platforms mobile phones are having rapid and great growth, currently it is not It has a universal and interoperable communication methodology that allows data exchange between a mobile web application and an application resident on a smart card.

Although APIs make it possible to read or write data in the memory of cards in a specific format (NDEF), it is not feasible to open interaction, execution of native commands, etc. about her.

The present work proposes the development of a protocol for communication and a smart card application, which using of existing APIs, remove restrictions and enable standard, low-level communication with the card and can use of all its computing capabilities.

Keywords: Smart card, NDEF, NFC

Índice general

Índice de Figuras **VI**

Índice de Abreviaciones **VII**

Índice de Tablas **XI**

1. Introducción **1**
1.1. Objetivos . . . 2
1.2. Estructura . . . 3

2. Estado del Arte **4**
2.1. Tecnología de tarjeta inteligente . . . 4
2.1.1. Tipos de tarjeta inteligente . . . 5
2.1.2. Características físicas . . . 5
2.1.3. Arquitectura . . . 7
2.1.4. Protocolos de comunicación . . . 8
2.1.5. Sistema operativo . . . 10
2.1.6. JavaCard . . . 11
2.1.7. Aplicación JavaCard . . . 11
2.1.8. Entorno de ejecución . . . 14
2.2. Tecnologías de comunicación de campo cercano . . . 16
2.2.1. Historia y evolución de NFC . . . 16
2.2.2. Principios de operación de NFC . . . 17
2.2.3. Especificaciones técnicas . . . 18
2.2.4. Mecanismos de seguridad en NFC . . . 19
2.2.5. NFC Data Exchange Format (NDEF) . . . 20
2.2.6. Comunicación mediante NFC . . . 22

3. Emulación de un tag NFC **24**
3.1. Requisitos funcionales . . . 24
3.2. Especificación funcional . . . 25

3.2.1. Estructura de memoria . . . 25
3.2.2. Comandos de operación . . . 28
3.3. Implementación del *applet* . . . 29
3.3.1. Manejo de la comunicación APDU . . . 32
3.4. Pruebas en dispositivos reales . . . 33

4. Protocolo de encapsulación de APDU sobre NDEF **39**
4.1. Descripción del protocolo . . . 40
4.1.1. Estructura de los mensajes NDEF . . . 40
4.1.2. Flujo de comunicación . . . 41
4.2. Implementación del protocolo . . . 42
4.3. Verificación en entorno real . . . 45

5. Aplicación web de validación **53**
5.1. Modificaciones al *applet* . . . 53
5.2. API `NDEFReader` . . . 54
5.2.1. Compatibilidad . . . 55
5.3. Desarrollo web . . . 56
5.3.1. Integración de la API `NDEFReader` . . . 57
5.4. Pruebas y validaciones . . . 59

6. Conclusiones **63**
6.1. Análisis de los objetivos alcanzados . . . 63
6.2. Mejoras y propuestas futuras . . . 64

Índice de figuras

2.1. Esquema de tipos de tarjetas inteligentes 5
2.2. Tarjeta inteligente genérica 6
2.3. Contactos del chip de una tarjeta inteligente 6
2.4. Arquitectura de tarjetas inteligentes 7
2.5. Tag NFC 16
2.6. Registro NDEF 21

3.1. Ejemplo de estructura de memoria de una etiqueta de tipo 4 26
3.2. Estructura del CC file 27
3.3. Estructura del mensaje NDEF 27
3.4. Ejemplo de implementación de memoria en el *applet* 31
3.5. Ejemplo de implementación de memoria en el *applet* 31
3.6. Ejemplo de implementación de CC File en el *applet* 31
3.7. Secuencia de lectura de un mensaje NDEF 33
3.8. Prueba de lectura desde móvil del *applet* de emulación de un tag. 34
3.9. Etiqueta por defecto instalada en el *applet* 34
3.10. Prueba de escritura desde móvil del *applet* de emulación de un tag 35
3.11. Secuencia de mensajes para la escritura de una etiqueta 36
3.12. Prueba de verificación de escritura del contenido en la etiqueta 37

4.1. Flujo de comunicación 42
4.2. Función de procesado de APDU encapsuladas 45
4.3. Prueba de funcionamiento del algoritmo usando PcscScriptor 46
4.4. Estado inicial del *applet* 47
4.5. Envió de primera APDU con comando usando NDEF 48
4.6. Lectura de respuesta de ejecución a la APDU enviada 49
4.7. Envió de segundo APDU solicitando lectura de APDU anterior usando NDEF 50
4.8. Recepción de la respuesta generada por la APDU encapsulada en un mensaje NDEF type `application/apdu-res` 51
4.9. Verificación de correcta escritura de otro tipo de mensajes NDEF 52

5.1. Compatibilidad de la API `NDEFReader` 56
5.2. Estructura de la web implementada 57
5.3. Método de codificación de una APDU en la web 58
5.4. Acceso a la web por primera vez 59
5.5. Confirmación de escritura en la web 60
5.6. Lectura desde la web 60
5.7. Etiqueta de resultado desde NFC Reader 61
5.8. Resultado de ejecución de la suma desde un ordenador usando PcscScriptor 62

Glosario

AES Advanced Encryption Standard

AID Application Identifier

APDU Application Protocol Data Unit

API Application Programming Interface

ASCII American Standard Code for Information Interchange

ASK Amplitude Shift Keying

ATR Answer To Reset

CAP Card Application File

CC File Card Capability File

CF Chunk Flag

CPU Central Processing Unit

CSS Cascading Style Sheets

DF Dedicated File

EEPROM Electrically Erasable Programmable Read-Only Memory

EF Elementary File

FID File Identifier

GSM Global System for Mobile Communications

HTML HyperText Markup Language

IEC International Electrotechnical Commission

IL Id Length

IoT Internet of Things

ISO International Organization for Standardization

JCRE Java Card Runtime Environment

JCVM Java Card Virtual Machine

LLCP Logical Link Control Protocol

MB Message Begin

ME Message End

MF Master File

MIME Multipurpose Internet Mail Extensions

NDEF NFC Data Exchange Format

NFC Near Field Communication

OOP Object-Oriented Programming

OTP One-Time Password

PIN Personal Identification Number

PIX Personal Identification Exchange

PM Phase Modulation

RAM Random Access Memory

RF Radio Frequency

RFID Radio Frequency Identification

RID Record Identifier

ROM Read-Only Memory

SMS Short Message Service

SNEP Simple NDEF Exchange Protocol

SR Short Record

SW Status Word

TDMA Time Division Multiple Access

TFG Trabajo de Fin de Grado

TLV Type-Length-Value

TNF Type Name Format

TPDU Transport Protocol Data Unit

URI Uniform Resource Identifier

Índice de tablas

2.1. Formato de APDU de comando 8
2.2. Formato de APDU de respuesta 9
2.3. Formato de Application Identifier (AID) 15

3.1. Formato del comando APDU de selección 28
3.2. Formato del comando APDU de lectura 29
3.3. Formato del comando APDU de escritura 29

4.1. Formato genérico para registro NDEF de tipo `application/apdu-req` . . 40
4.2. Formato genérico para registro NDEF de tipo `application/apdu-res` . . 40

5.1. Formato de APDU de comando con valores de ejemplo 53

Capítulo 1

Introducción

En las últimas décadas, las tarjetas inteligentes han emergido como una herramienta clave en una amplia gama de aplicaciones, desde sistemas de pago hasta identificación, transporte y acceso seguro. Estas tarjetas, equipadas con un chip que permite almacenar y procesar datos de manera autónoma, han transformado la forma en que interactuamos con sistemas digitales y físicos. Su capacidad para garantizar la seguridad de la información y realizar transacciones rápidas y fiables las ha hecho indispensables en sectores como la banca, los servicios gubernamentales y las telecomunicaciones.

A medida que la tecnología ha avanzado, estas tarjetas han evolucionado hacia sistemas sin contacto, una solución que ha revolucionado la interacción entre el usuario y los dispositivos. Estos sistemas sin contacto, basados en tecnologías como Near Field Communication (NFC), permiten una interacción aún más rápida y sencilla, eliminando la necesidad de contacto físico entre la tarjeta y el lector. Así, la evolución de las tarjetas inteligentes ha facilitado un cambio hacia interacciones más fluidas, seguras y convenientes, tanto en transacciones diarias como en el acceso a servicios esenciales.

Sin embargo, la evolución hacia un mundo sin contacto ha traído consigo nuevos desafíos, especialmente en lo que respecta a la seguridad y la interoperabilidad de los sistemas. A medida que los entornos digitales se vuelven más cerrados y especializados, surge la necesidad de establecer protocolos que permitan una comunicación fluida y segura entre los distintos dispositivos y plataformas. Este reto es particularmente evidente en el uso de tarjetas inteligentes, las cuales son ampliamente utilizadas en diversos sectores debido a su capacidad para almacenar información de manera segura y su resistencia a manipulaciones.

A pesar de sus ventajas, las tarjetas inteligentes enfrentan limitaciones cuando se trata de integrarse con entornos más abiertos, como los navegadores web. La falta de un

protocolo estándar que permita la comunicación directa y segura entre estas tarjetas y las aplicaciones web ha generado una barrera significativa para su adopción en nuevos contextos digitales. Este problema se ve agravado por la naturaleza cerrada de muchos entornos de acceso, lo que dificulta la creación de soluciones universales y accesibles.

Ante esta situación, parece fundamental explorar métodos más abiertos y flexibles para integrar las tarjetas inteligentes en el ecosistema digital actual. Uno de los enfoques más prometedores es la utilización del formato de intercambio de datos NFC Data Exchange Format (NDEF), que permite la comunicación entre dispositivos NFC y aplicaciones a través de un protocolo estandarizado. Este Trabajo de Fin de Grado (TFG) se centrará en la definición y desarrollo de un ecosistema que permita la interacción eficiente y segura entre las tarjetas inteligentes y las aplicaciones web utilizando NDEF, con el objetivo de superar las barreras actuales y facilitar un mayor uso de estas tecnologías en diversos contextos digitales.

Los navegadores y los dispositivos móviles únicamente tienen la capacidad nativa de comunicarse con etiquetas NFC, que en esencia son tarjetas. Sin embargo, el problema es que solo se utiliza su capacidad de almacenamiento de información, sin poder acceder a las funcionalidades más avanzadas que si poseen las tarjetas inteligentes. Por esta razón, buscamos definir un protocolo que permita aprovechar esas capacidades adicionales a través de este canal.

1.1. Objetivos

A partir del contexto descrito, los objetivos del proyecto se amplían para abordar de manera integral los retos y oportunidades que presentan las tecnologías sin contacto, especialmente en la integración de tarjetas inteligentes con entornos web y dispositivos móviles. Los objetivos específicos del proyecto son:

- **Explorar las funcionalidades nativas de dispositivos móviles y navegadores web para la comunicación NFC:** Analizar en detalle las capacidades actuales que tienen los dispositivos móviles y navegadores para interactuar con etiquetas NFC, evaluando su potencial para comunicarse con tarjetas inteligentes.
- **Definir un protocolo de comunicación basado en NDEF para tarjetas inteligentes:** Desarrollar un protocolo que utilice el formato de intercambio de datos NDEF, permitiendo una interacción segura y eficiente entre tarjetas inteligentes y aplicaciones web a través de dispositivos móviles y navegadores. Este protocolo deberá considerar aspectos críticos como la seguridad, interoperabilidad y la experiencia de usuario.

- **Implementar un prototipo funcional para validar la solución propuesta:** Construir un prototipo que demuestre la viabilidad de la solución desarrollada. Este prototipo deberá permitir la lectura y escritura de datos entre tarjetas inteligentes y aplicaciones web mediante NFC, evidenciando la posibilidad de superar las barreras actuales de interoperabilidad.
- **Evaluar la seguridad y rendimiento del sistema:** Realizar un análisis del sistema en términos de seguridad y rendimiento, evaluando su resistencia a posibles vulnerabilidades y su capacidad para operar de manera eficiente en escenarios reales.
- **Promover la adopción de la solución en diferentes sectores:** Identificar sectores específicos donde la solución desarrollada pueda tener un impacto significativo, como en el sector bancario, gubernamental, salud, entre otros, y proponer estrategias para su adopción y despliegue a gran escala.

1.2. Estructura

La estructura de la memoria comienza con un capítulo de Introducción, que establece el contexto y los objetivos del proyecto en relación con la tecnología sin contacto y la integración de tarjetas inteligentes en entornos web. Este capítulo prepara el terreno para entender la importancia y los desafíos de la tecnología NFC.

A continuación, se aborda el Estado del Arte, que proporciona una base teórica sobre la evolución y la relevancia de la tecnología en el ámbito tecnológico, estableciendo un marco para el proyecto. Seguidamente, se detalla el desarrollo del primer Applet de Emulación de una etiqueta de tipo 4, describiendo minuciosamente el proceso de diseño, programación y los desafíos técnicos superados. Tras esto, se explora el Protocolo de Comunicación NDEF, explicando su funcionamiento y su papel en la transmisión de datos mediante tecnología NFC.

La memoria avanza con una Demo de Funcionamiento, donde se presenta de forma práctica cómo la solución desarrollada interactúa con dispositivos compatibles, demostrando su funcionalidad. Finalmente, el capítulo de Conclusiones ofrece una reflexión sobre los resultados obtenidos, destacando las contribuciones del proyecto y sugiriendo posibles mejoras y aplicaciones futuras.

Capítulo 2

Estado del Arte

2.1. Tecnología de tarjeta inteligente

Las tarjetas inteligentes han evolucionado significativamente desde la introducción de las primeras tarjetas plásticas a mediados del siglo XX. Aunque en apariencia se asemejan a una tarjeta de crédito convencional, su funcionamiento es muy distinto. Estas tarjetas están equipadas con un circuito integrado que les permite almacenar, proteger y procesar datos, lo que las convierte en pequeños ordenadores portátiles. A diferencia de las antiguas tarjetas con banda magnética, las tarjetas inteligentes ofrecen una mayor seguridad y funcionalidad gracias a este chip.

El concepto de la tarjeta con chip integrado surgió a principios de la década de 1970, cuando un ingeniero japonés presentó un prototipo que incorporaba un circuito integrado en una tarjeta plástica. Aunque este dispositivo inicial solo servía como almacenamiento de memoria, en 1974 el francés Roland Moreno desarrolló lo que hoy reconocemos como la primera tarjeta inteligente. Esta innovadora tarjeta introdujo mecanismos de seguridad, como el uso de un Personal Identification Number (PIN), el cual sigue siendo un estándar en la actualidad[1]. Sin embargo, no fue hasta la década de 1980 que las tarjetas inteligentes comenzaron a comercializarse con éxito, resolviendo problemas como el fraude bancario y las falsificaciones en el ámbito de las tarjetas de crédito y prepago.

Los avances en la tecnología de circuitos integrados y criptografía han permitido que las tarjetas inteligentes se conviertan en dispositivos más poderosos, seguros y accesibles. Su capacidad para almacenar y procesar información confidencial las hace ideales para aplicaciones que requieren alta seguridad, como la generación y custodia de claves secretas y la ejecución de algoritmos criptográficos. Además de su uso en la banca y la telefonía, las tarjetas inteligentes se emplean como llaves de acceso, identificadores en teléfonos móviles y en una amplia variedad de otros servicios.

2.1.1. Tipos de tarjeta inteligente

Las tarjetas inteligentes, o *smart cards*, se clasifican principalmente según dos criterios los cuales se pueden apreciar en la figura 2.1.

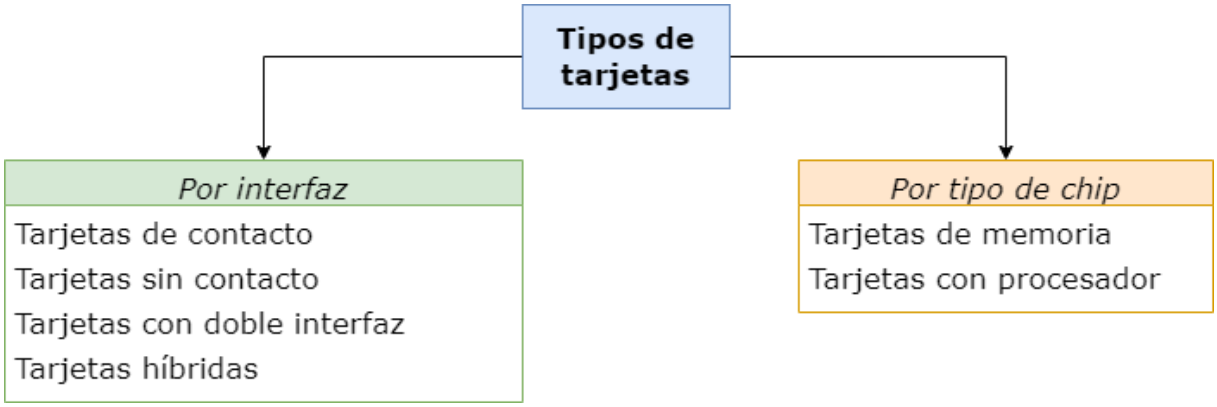

Figura 2.1: Esquema de tipos de tarjetas inteligentes

En cuanto a la interfaz, las tarjetas de contacto requieren ser insertadas físicamente en un lector para que se pueda realizar la transferencia de datos. Por otro lado, las tarjetas sin contacto emplean tecnología Radio Frequency Identification (RFID) [2], permitiendo la comunicación con el lector sin contacto físico. Las tarjetas de doble interfaz combinan ambas modalidades en un solo chip, lo que las hace versátiles para aplicaciones que requieran tanto contacto como la facilidad del uso sin contacto. Las tarjetas híbridas, aunque también ofrecen ambas modalidades, lo hacen a través de dos chips separados, lo que puede generar problemas de sincronización de la información entre los dos sistemas, complicando su uso en entornos que requieren un manejo más coherente y seguro de los datos.

En cuanto al tipo de chip, las tarjetas de memoria se limitan a almacenar, leer y escribir datos, pero no son capaces de procesar información. Esto las convierte en una opción económica y adecuada para aplicaciones más simples, como el control de acceso. Las tarjetas con microprocesador, en cambio, cuentan con un chip capaz de procesar datos además de almacenarlos, lo que las hace comparables a pequeñas computadoras. Estas tarjetas son empleadas en aplicaciones que requieren mayor seguridad y capacidad de procesamiento, como los pagos electrónicos o la autenticación de identidad.

2.1.2. Características físicas

Las tarjetas inteligentes con microprocesador son dispositivos que combinan funcionalidades avanzadas de almacenamiento y procesamiento en un formato compacto. Desde

un punto de vista físico (ver figura 2.2), estas tarjetas están diseñadas bajo estándares internacionales que aseguran su interoperabilidad, durabilidad y facilidad de uso. Dos de los formatos más comunes definidos por la norma International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 7810 [3] son el ID-1 y el ID-000, cada uno con aplicaciones específicas.

Figura 2.2: Tarjeta inteligente genérica

En cuanto a sus materiales, las tarjetas inteligentes generalmente están hechas de PVC, pero también pueden usar materiales como PET, policarbonato, o combinaciones de estos para mejorar la resistencia al desgaste, la flexibilidad y la durabilidad. Estas tarjetas deben soportar condiciones físicas difíciles, incluyendo variaciones de temperatura, exposición a humedad, y manejo constante.

Otra de las características mas notables son los pines metálicos en la superficie de la tarjeta, que permiten la comunicación con el lector. Según la norma ISO/IEC 7816 [4], las tarjetas de contacto tienen ocho pines o contactos, cada uno con una función específica. La figura 2.3 respresenta la disposición de los estos contactos metálicos.

Figura 2.3: Contactos del chip de una tarjeta inteligente

- **Vcc:** Alimenta el microprocesador con un voltaje típico de 5V, aunque también puede operar a 3V o 1.8V en modelos más recientes.
- **RST:** Línea de reinicio que se utiliza para inicializar o reiniciar el microprocesador.
- **CLK:** Proporciona la señal de reloj al microprocesador para sincronizar las operaciones.
- **RFU:** Reservado para usos futuros (no utilizado en la mayoría de las tarjetas).
- **GND:** Conexión a tierra que cierra el circuito eléctrico.
- **Vpp:** Inicialmente usado para proporcionar el voltaje de programación para la memoria Electrically Erasable Programmable Read-Only Memory (EEPROM). Hoy en día está en desuso en la mayoría de las tarjetas modernas.
- **I/O:** Línea de entrada/salida utilizada para la transmisión bidireccional de datos entre la tarjeta y el lector.
- **RFU:** También reservado para usos futuros, aunque en algunas aplicaciones específicas puede tener uso asignado.

2.1.3. Arquitectura

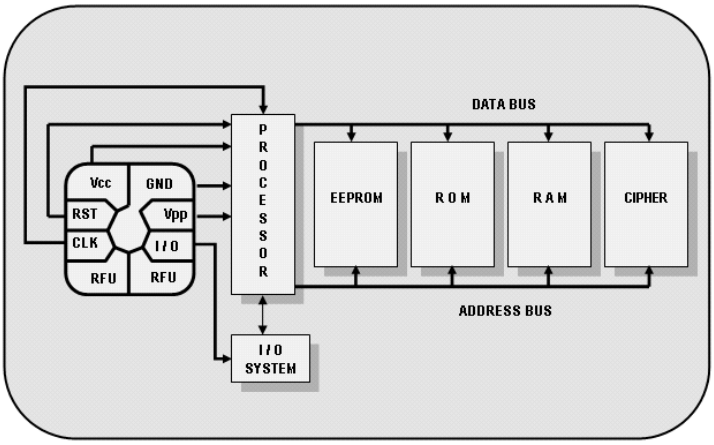

Figura 2.4: Arquitectura de tarjetas inteligentes

Las tarjetas inteligentes suelen adoptar una arquitectura basada en el modelo de la máquina de Von Neumann, la cual se caracteriza por tener una Central Processing Unit (CPU) que ejecuta instrucciones de manera secuencial, almacenadas junto con los datos en una memoria unificada. En el contexto de las tarjetas inteligentes, el microcontrolador

incluye tanto el microprocesador como tres tipos de memoria: Read-Only Memory (ROM), EEPROM y Random Access Memory (RAM). Estas componentes están integradas en un solo chip, lo que hace costosa y difícil la intercepción de las señales entre el procesador y la memoria, garantizando una alta seguridad física para los datos almacenados. Esta estructura se puede apreciar de manera resumida en la figura 2.4

2.1.4. Protocolos de comunicación

La comunicación entre un dispositivo y una tarjeta inteligente siempre se inicia desde el aparato externo, estableciendo una relación maestro-esclavo en la que el terminal actúa como el maestro y la tarjeta como el esclavo. Esta comunicación se realiza utilizando un modelo serie asíncrono en un canal **half-duplex**, en el cual los dispositivos deben turnarse para transmitir datos, ya que solo existe un canal entre ellos. A pesar de que la mayoría de los microprocesadores modernos incluyen canales separados de entrada y salida, el protocolo **full-duplex** no está disponible en la actualidad.

2.1.4.1. Intercambio de información: APDU's

El intercambio de información entre el dispositivo externo y la tarjeta se lleva a cabo mediante unidades de datos denominadas Application Protocol Data Unit (APDU), cuya estructura está especificada en la norma ISO 7816-4. Una APDU puede ser de dos tipos: **comando**, que transfiere datos desde la aplicación externa a la tarjeta, y **respuesta**, enviada desde la tarjeta hacia el lector.

Cada vez que se inserta o acerca una tarjeta al lector, se realiza un proceso de activación, transmitiendola la energía suficiente para funcionar. La tarjeta responde con un mensaje Answer To Reset (ATR), que contiene información crucial sobre cómo se debe llevar a cabo la comunicación, incluyendo la estructura de los datos y el protocolo de transmisión. Una vez que el lector interpreta el ATR, se envía la primera instrucción en forma de APDU, y la tarjeta responde con los datos solicitados o con un código de estado.

2.1.4.2. Estructura de una APDU de comando

En la tabla 2.1 se puede observar la estructura de una APDU la cual tiene los siguientes campos:

CLA	INS	P1	P2	Lc	Datos Opcionales	Le

Tabla 2.1: Formato de APDU de comando

- **CLA**: Un byte que indica la clase de la instrucción (por ejemplo, tipo de seguridad).
- **INS**: Un byte que determina el comando enviado.
- **P1** y **P2**: Dos bytes que actúan como parámetros específicos del comando.
- **Lc**: Un byte que especifica la longitud de los datos enviados, si los hubiera.
- **Datos opcionales**: Una serie de bytes con longitud variable.
- **Le**: Un byte que especifica la longitud esperada de los datos en la APDU de respuesta.

2.1.4.3. Estructura de una APDU de respuesta

En la tabla 2.2 se puede observar la estructura de una APDU de respuesta la cual esta formada por los siguientes campos:

Datos Opcionales	SW1	SW2

Tabla 2.2: Formato de APDU de respuesta

- **Datos opcionales**: Contiene la respuesta al comando enviado.
- **SW1 y SW2**: Dos bytes conocidos como *Status Word*, que indican el resultado de la operación. Por ejemplo, el valor hexadecimal `90 00` indica que el comando se ejecutó con éxito.

2.1.4.4. Protocolos de transmisión T=0 y T=1

Las APDU se transmiten utilizando un protocolo de transmisión de nivel inferior mediante Transport Protocol Data Unit (TPDU). La norma ISO 7816-3 [5] especifica varios protocolos de transmisión, de los cuales los más comunes son **T=0** y **T=1**. Ambos emplean un modelo de transmisión asíncrono y half-duplex. El protocolo T=0 está orientado a bytes, donde cada byte se transmite de forma independiente, utilizando un bit de paridad para la detección de errores. En caso de error, el byte se retransmite. Este protocolo es simple y es utilizado en aplicaciones como monederos electrónicos y Global System for Mobile Communications (GSM).

Por otro lado, el protocolo T=1 está orientado a bloques, permitiendo un control de flujo en ambas direcciones, lo que posibilita que la tarjeta asuma el control de la comunicación. Aunque los lectores soportan ambos protocolos, el que se utiliza en la comunicación depende del ATR recibido inicialmente.

2.1.5. Sistema operativo

El sistema operativo de una tarjeta inteligente está diseñado principalmente para gestionar un sistema de archivos estructurado en un árbol jerárquico. Este árbol se organiza en diferentes tipos de archivos: el Master File (MF), los Dedicated File (DF) y los Elementary File (EF). El MF es la raíz del sistema de archivos, mientras que los DF son subdirectorios que agrupan diferentes EF, los cuales contienen los datos o configuraciones almacenados en la tarjeta.

A lo largo del tiempo, estos sistemas operativos han evolucionado para adaptarse a las crecientes demandas de seguridad y funcionalidad. Aunque operan bajo condiciones de memoria y procesamiento muy limitadas, han incorporado funciones orientadas a la gestión de la memoria, la seguridad y la comunicación con el exterior a través de la APDU, estandarizada por ISO 7816-4 [1].

2.1.5.1. Sistemas operativos modernos en tarjetas inteligentes

Los sistemas operativos modernos para tarjetas inteligentes han evolucionado significativamente desde los primeros sistemas basados en la gestión simple de archivos. Estos sistemas operativos ahora soportan modelos multinivel y multicapa, permitiendo una mayor flexibilidad, seguridad y capacidad de actualización. Algunos de los sistemas operativos más utilizados en la actualidad son los siguientes:

- **JavaCard [6]:** Es uno de los sistemas operativos más conocidos y ampliamente utilizado en tarjetas inteligentes. Basado en el lenguaje de programación Java, proporciona un entorno seguro y portátil para el desarrollo de aplicaciones. Su arquitectura permite la ejecución de múltiples aplicaciones en una sola tarjeta, gestionando recursos de manera segura y compatible con los estándares ISO 7816.
- **MULTOS [7]:** MULTOS es un sistema operativo basado en un entorno de ejecución de aplicaciones seguras, que también es compatible con estándares internacionales. Su diseño modular permite la instalación de diferentes aplicaciones sobre una sola tarjeta sin comprometer la seguridad de las otras. MULTOS es conocido por su robustez y se utiliza en una variedad de aplicaciones.
- **CardOS [8]:** Desarrollado por Giesecke+Devrient, CardOS es un sistema operativo altamente seguro que se emplea en aplicaciones financieras y de identificación. CardOS soporta aplicaciones tanto de alto como de bajo nivel, y se caracteriza por su capacidad para ofrecer un alto grado de seguridad, así como por su flexibilidad para adaptarse a las diferentes necesidades del mercado.

- **ISO/IEC 7816-4 Compliance Systems:** Algunos sistemas operativos son diseñados para cumplir estrictamente con la norma ISO/IEC 7816-4, proporcionando una estructura y funcionalidad básica, pero con soporte para una amplia variedad de aplicaciones. Estos sistemas se centran en la compatibilidad y la interoperabilidad entre diferentes fabricantes y aplicaciones.

2.1.6. JavaCard

JavaCard ofrece una Application Programming Interface (API) estandarizada compuesta por varios paquetes clave para facilitar el desarrollo de aplicaciones en tarjetas inteligentes. El paquete `java.lang` define las clases fundamentales y las excepciones básicas necesarias para el desarrollo [9]. El paquete `javacard.framework` proporciona herramientas esenciales para gestionar *applets* y comunicarse con el terminal, cumpliendo con la norma ISO 7816-4 [10]. Los paquetes `javacard.security` y `javacardx.crypto` ofrecen funciones para implementar algoritmos criptográficos y garantizar la seguridad en las transacciones [11]. Finalmente, el paquete java.io ofrece funcionalidades adaptadas para la manipulación de flujos de entrada y salida, considerando las limitaciones de la plataforma. El empleo de estos paquetes asegura una interfaz estándar para los desarrolladores y mejorando el rendimiento al optimizar las aplicaciones para el hardware particular de cada tarjeta, al ser codificados de forma nativa para el procesador específico.

A pesar de estas ventajas, JavaCard presenta ciertas limitaciones en comparación con Java estándar. La plataforma admite solo tipos de datos primitivos básicos como `boolean`, `byte`, `short` y, opcionalmente, `int`, excluyendo tipos más grandes como `long`, `float` o `double`. Los arrays unidimensionales son soportados, pero los arrays multidimensionales y estructuras de datos complejas no están disponibles. Aunque JavaCard sigue los principios básicos de la Object-Oriented Programming (OOP), carece de características avanzadas como la herencia múltiple y la reflexión.

El manejo de excepciones está disponible pero con un conjunto más limitado, lo que requiere una gestión precisa de los errores para evitar comprometer los recursos finitos de la tarjeta. Además, JavaCard no garantiza un recolector de basura, obligando a los desarrolladores a gestionar manualmente los recursos de memoria para evitar problemas de fragmentación o agotamiento.

2.1.7. Aplicación JavaCard

Todos los *applets* deben extender la clase abstracta `javacard.framework.Applet` que contiene los métodos que gestionan el ciclo de vida del *applet*. A pesar de ser una clase abstracta, sus miembros no lo son y no se requiere re-implementarlos. No obstante, algu-

nos de ellos como los que controlan el proceso de instalación o el manejo de las diferentes APDU, sí es recomendable que se implementen para una mejor interacción con el JCRE.

El ciclo de vida de un *applet* se corresponde con los diferentes métodos que define la clase Applet:

1. Diseño, implementación y compilación del *applet*
2. Instalación y registro — `constructor()` / `install()`
   Instancia e inicializa los objetos definidos en el applet, al tiempo que realiza el registro del *applet* en el JCRE.
3. Selección del *applet* — `select()`
   Activa del *applet* para la recepción de APDU, es decir, toda APDU que no puede ser interpretada por el JCRE se remite al *applet*.
4. Intercambio de APDU — `process()`
   Gestiona todos los comandos APDU recibidos y se puede considerar que contiene la lógica del *applet*.
5. Deselección del *applet* — `deselect()`
   Informa que otro *applet* ha sido seleccionado.
6. Desactivación de la tarjeta
   Implica abortar transacciones y el borrado de objetos temporales.

La función `install()` (ejemplo en algoritmo 1) se encarga de la instalación y configuración inicial de la aplicación en la tarjeta inteligente. Esta función se ejecuta una sola vez durante el proceso de carga de la aplicación en la tarjeta y es responsable de la asignación de recursos necesarios, como el almacenamiento de datos y la configuración de las estructuras internas. `install()` establece el entorno en el que la aplicación JavaCard funcionará, preparando la tarjeta para recibir y procesar comandos APDU de manera eficiente.

Algoritmo 1 Método install

```
1: Método install(byte[] bArray, short bOffset, byte bLength)
2: Inicializar la aplicación
3: Guardar el AID y otros datos necesarios
4: RegistrarApplet(AID)
```

Otra función importante es la gestión de la selección de la aplicación, realizada a través de la función `select()`. Este método se invoca cuando una aplicación en la tarjeta es seleccionada por el lector. La función `select()` permite a la aplicación establecer el

contexto inicial para el manejo de comandos posteriores, asegurando que la tarjeta sepa cuál aplicación debe manejar el comando APDU. Además, la función `select()` puede inicializar recursos específicos de la aplicación o configurar parámetros necesarios para el funcionamiento correcto de la misma. En el algoritmo 2 se muestra un resumen de las funcionalidades básicas integradas en este método.

Algoritmo 2 Método select

```
1: Método select()
2: Establecer el contexto de la aplicación
```

Otra de las funciones clave en una aplicación JavaCard es el método `process()`, que actúa como el núcleo del procesamiento de comandos. Este método recibe las solicitudes APDU enviadas desde el lector de tarjetas y es responsable de interpretar y ejecutar las operaciones solicitadas. Dependiendo del comando recibido, `process()` puede realizar una variedad de tareas, como leer o escribir datos en la tarjeta, autenticar al usuario o realizar cálculos específicos. En el algoritmo 3 se muestra un pseudocódigo representativo de la funcionalidad descrita.

Algoritmo 3 Método process

```
1: Método process(APDU apdu)
2: if apdu.isValid() then
3:     Leer el comando APDU
4:     byte comando = apdu.getCommand()
5:     if comando = 0x00 then
6:         Ejecutar lógica de lectura
7:         byte[] datos = LeerDatos()
8:         apdu.setResponseData(datos)
9:     else if comando = 0x01 then
10:        Ejecutar lógica de escritura
11:        byte[] datosEntrada = apdu.getData()
12:        EscribirDatos(datosEntrada)
13:    else
14:        Comando no reconocido
15:        apdu.setResponseData("Error: Comando no reconocido".getBytes())
16:    end if
17: else
18:    Comando inválido
19:    apdu.setResponseData("Error: Comando inválido".getBytes())
20: end if
```

2.1.8. Entorno de ejecución

La gestión en tiempo de ejecución de los recursos de una tarjeta Java Card es realizada por el Java Card Runtime Environment (JCRE) [12]. El JCRE actúa como el sistema operativo de la tarjeta, encargándose de la gestión de recursos, comunicaciones, ejecución de applets, y seguridad.

En una tarjeta Java Card, el JCRE está activo de forma permanente. Su ciclo de vida comienza en la fase de personalización de la tarjeta, cuando se graba en la ROM. Durante este proceso, se inicializa la Java Card Virtual Machine (JCVM) y se crean los objetos necesarios para soportar los servicios del JCRE y la gestión de los applets. Estos objetos se crean en memoria persistente (EEPROM) para conservar la información incluso cuando la tarjeta no está alimentada.

El proceso de ejecución y gestión de *applets* en una tarjeta Java Card sigue los siguientes pasos:

- **Recepción de comandos APDU:** El JCRE entra en un ciclo de espera a la recepción de comandos APDU a través de su interfaz de comunicaciones. Al recibir un comando, el JCRE selecciona un *applet* o redirige el comando a un *applet* previamente seleccionado para que procese la APDU. El *applet* envía la respuesta mediante otra APDU, y el JCRE vuelve a esperar nuevos comandos.
- **Persistencia y Transitoriedad de Objetos:** Por defecto, los objetos en Java Card se alojan en memoria persistente. Sin embargo, los *applets* pueden crear objetos transitorios en memoria RAM, los cuales se eliminan al desconectar la alimentación de la tarjeta.
- **Transacciones Atómicas:** Para evitar estados inconsistentes debido a posibles cortes de alimentación, las estructuras internas de los *applets* se modifican de forma atómica. Java Card proporciona un conjunto de funciones en su API que permiten definir la ejecución atómica de un conjunto de operaciones, garantizando que una transacción se realice completamente o se vuelva al estado previo.
- **Mecanismos de Seguridad:** Java Card implementa firewalls para evitar que las acciones de un *applet* afecten a otros. Además, permite la compartición de datos y servicios de forma segura entre *applets* cuando sea necesario.

Además, los paquetes y aplicaciones Java Card se identifican de manera única mediante un Application Identifier (AID), que se compone de un Record Identifier (RID) de 5 bytes y un Personal Identification Exchange (PIX) de hasta 11 bytes como se puede observar en la tabla 2.3.

RID (5 Bytes)	PIX (0-11 Bytes)

Tabla 2.3: Formato de AID

Al volver a alimentar la tarjeta, el JCRE arranca la JCVM, cargando los datos previamente almacenados en memoria persistente y reanudando su operación.

2.1.8.1. Compilación y proceso de instalación en la tarjeta

Tras desarrollar el código en el subconjunto de Java compatible con JavaCard, el siguiente paso es la compilación y preparación del código para su instalación en la tarjeta inteligente. Este proceso incluye varias etapas críticas:

- **Conversión a formato Card Application File (CAP):** El código Java es transformado en un formato específico de JavaCard llamado CAP. Este formato es más compacto y adecuado para las limitaciones de almacenamiento y procesamiento de las tarjetas inteligentes.
- **Proceso de Carga e Instalación:** El archivo CAP se transfiere a la tarjeta inteligente, donde es verificado para asegurar su integridad. Posteriormente, se carga en la tarjeta y se enlaza con las clases existentes en un entorno seguro que garantiza la correcta instalación y operación del código.
- **Ejecución Segura:** Durante la ejecución, la JCVM traduce el bytecode en instrucciones nativas del procesador de la tarjeta. Este proceso se realiza bajo un estricto control para prevenir violaciones de seguridad y garantizar un uso eficiente de los limitados recursos de la tarjeta.

Global Platform [13] es un conjunto de estándares y especificaciones diseñado para la gestión segura de aplicaciones en tarjetas inteligentes, como las tarjetas JavaCard. Este marco común asegura la interoperabilidad entre diferentes tarjetas, lectores y sistemas de backend, estableciendo reglas para la instalación, administración y eliminación de aplicaciones de manera segura. Además, protege tanto los datos como las aplicaciones almacenadas en las tarjetas. Global Platform define un conjunto de comandos APDU que permiten la carga, instalación y gestión de aplicaciones en las tarjetas inteligentes. Los actores que participan en este ecosistema incluyen emisores de tarjetas, proveedores de software y hardware, fabricantes de tarjetas y autoridades de certificación, todos comprometidos con la seguridad y confiabilidad del sistema.

2.2. Tecnologías de comunicación de campo cercano

La tecnología de Comunicación de Campo Cercano (NFC, por sus siglas en inglés) [14] es una forma de comunicación inalámbrica de corto alcance que permite la transferencia de datos entre dispositivos cuando están a pocos centímetros de distancia. Esta tecnología se basa en la inducción de campo electromagnético para permitir la comunicación entre un dispositivo lector y un dispositivo objetivo, generalmente sin la necesidad de una fuente de energía propia en el dispositivo objetivo, del cual tenemos un ejemplo en la 2.5.

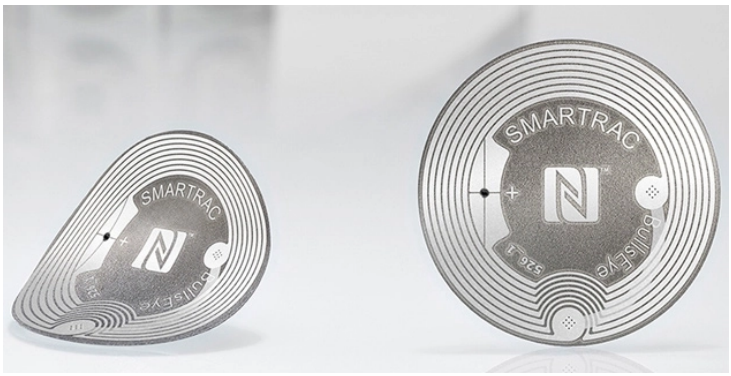

Figura 2.5: Tag NFC

2.2.1. Historia y evolución de NFC

Los orígenes de la tecnología NFC se remontan a los desarrollos en tecnologías de comunicación inalámbrica y tarjetas inteligentes en las décadas de 1980 y 1990. La tecnología se basa en conceptos preexistentes de RFID, que permitían la comunicación de datos a corta distancia mediante campos electromagnéticos. A medida que estas tecnologías evolucionaron, surgió la necesidad de un método más versátil y estandarizado para la comunicación cercana, lo que llevó al desarrollo de NFC.

A principios de los años 2000, Nokia, Philips y Sony colaboraron para estandarizar la tecnología NFC. En 2004, formaron el Foro NFC para impulsar su adopción y desarrollar especificaciones uniformes. No fue hasta el 2006 que se publico el primer estandar de NFC, dando esto lugar al establecimiento de la tecnologia en el mercado.

Uno de los hitos significativos fue el lanzamiento del primer teléfono móvil con NFC, el Nokia 6131, en 2006. A raiz de estos logros, esta tecnología se incorporó en diversas aplicaciones, incluyendo sistemas de pago móvil y control de acceso entre otros.

Desde su aparición, NFC ha experimentado una adopción creciente en diversas industrias. La evolución de NFC ha sido impulsada por la expansión de sus aplicaciones en pagos sin contacto y la integración con otros estándares tecnológicos como Bluetooth y Wi-Fi. En 2012, el uso de NFC en pagos móviles comenzó a despegar con la introducción de servicios como Google Wallet, y desde entonces, la gran mayoría de fabricantes han integrado NFC en sus dispositivos móviles.

La tecnología NFC sigue evolucionando, con mejoras en velocidad de transferencia y seguridad, y una mayor integración con sistemas de identificación y autentificación. Las tendencias actuales incluyen el uso de NFC en el Internet de las Cosas (Internet of Things (IoT)) y en aplicaciones de realidad aumentada, lo que sugiere que NFC seguirá desempeñando un papel importante en la comunicación inalámbrica y la interacción digital en el futuro.

2.2.2. Principios de operación de NFC

La tecnología NFC, como hemos introducido anteriormente, se basa en la transmisión de datos a través de campos electromagnéticos, concretamente en la banda de frecuencia de 13,56 MHz, utilizando la tecnología de acoplamiento inductivo para transferir información entre dispositivos ubicados a distancias muy cortas, típicamente en el rango de unos pocos centímetros. Este método permite una comunicación rápida y segura sin la necesidad de una alineación precisa de los dispositivos.

Esta tecnología admite tres modos de operación principales, cada uno diseñado para satisfacer diferentes necesidades de comunicación:

- **Modo lector/escritor:** En este modo, un dispositivo NFC actúa como lector o escritor y puede leer datos de o escribir datos en un dispositivo objetivo, como una etiqueta NFC. Las etiquetas NFC, que son pasivas y no requieren fuente de energía propia, contienen información que el lector puede acceder al acercar la etiqueta al lector.
- **Modo emulación de tarjeta:** En este modo, el dispositivo NFC se comporta como una tarjeta inteligente tradicional, permitiendo que se utilice en sistemas de pago o control de acceso. El dispositivo emulador de tarjeta emite una señal que puede ser leída por un lector de tarjetas NFC, permitiendo transacciones o autenticación sin contacto físico.
- **Modo peer-to-peer:** Este modo permite que dos dispositivos NFC se comuniquen entre sí directamente. Ambos dispositivos pueden actuar como iniciadores y objetivos, lo que permite el intercambio bidireccional de datos.

Cada uno de estos modos ofrece diferentes capacidades y aplicaciones, adaptándose a las diversas necesidades de comunicación. La flexibilidad de estos modos permite que NFC sea una tecnología versátil y eficiente para una amplia gama de usos.

2.2.3. Especificaciones técnicas

NFC está estandarizado por varias organizaciones, incluyendo la ISO 18092 [15] e ISO 21481 [15] y los ECMA 340 [16], ECMA 352 [17] y ECMA 356 [18]. Los protocolos principales que utiliza son el NFC-A (basado en la tecnología de tarjetas contactless tipo A de ISO/IEC 14443), NFC-B [19] (basado en tarjetas tipo B), y NFC-F [20] (basado en la tecnología FeliCa desarrollada por Sony). Estos estándares garantizan la compatibilidad entre dispositivos de diferentes fabricantes y aplicaciones.

La tecnología NFC opera en la banda de frecuencia de 13.56 MHz, una banda no licenciada dentro del espectro de Radio Frequency (RF) se utiliza globalmente para aplicaciones de comunicación de corto alcance. La distancia máxima de operación suele estar limitada a 10 centímetros, lo que añade un nivel adicional de seguridad al requerir proximidad física para la transferencia de datos. Esta limitación de distancia también minimiza el riesgo de interceptación no autorizada.

En cuanto a la velocidad de transferencia de datos, NFC soporta varias opciones, generalmente de 106, 212 y 424 kbps. Sin embargo, algunos dispositivos y estándares más recientes han comenzado a experimentar con velocidades más altas, como 848 kbps, para mejorar el rendimiento en aplicaciones que requieren un mayor ancho de banda. La elección de la velocidad depende del tipo de aplicación y de los dispositivos involucrados. Aunque estas velocidades son relativamente modestas en comparación con otras tecnologías inalámbricas, son adecuadas para aplicaciones típicas de NFC, como la autenticación, el intercambio de pequeños volúmenes de datos y los pagos.

Para la transmisión de datos, NFC emplea diferentes esquemas de modulación y codificación dependiendo del protocolo y la velocidad. Por ejemplo, NFC-A utiliza modulación de amplitud en un solo lado (Amplitude Shift Keying (ASK)) con una profundidad de modulación del 100 % y codificación Manchester a 106 kbit/s, mientras que NFC-F recurre a la codificación de variación de fase (Phase Modulation (PM)). La potencia de transmisión de NFC es limitada, generalmente en el rango de los miliwatts, lo que contribuye a su corto alcance. Además, NFC incluye mecanismos de detección de colisiones para evitar interferencias cuando varios dispositivos NFC están presentes en un entorno cercano.

La potencia de transmisión de NFC está restringida, típicamente en el rango de los miliwatts, lo que contribuye a su limitado alcance. Esta limitación en la potencia también

ayuda a reducir el riesgo de interferencias entre dispositivos. Además, NFC incorpora mecanismos diseñados para la detección y corrección de colisiones, asegurando una comunicación más eficiente en entornos donde varios dispositivos NFC están operando simultáneamente. Estos métodos son:

1. Detección de colisiones en modo pasivo: en el modo pasivo, el dispositivo activo (como un lector NFC) genera el campo electromagnético necesario para la comunicación, mientras que el dispositivo pasivo (como una etiqueta NFC) modula la señal para transmitir su respuesta. La detección de colisiones se basa en la observación de la respuesta del dispositivo pasivo. Si dos o más dispositivos pasivos responden simultáneamente, el dispositivo activo puede detectar una colisión porque la señal recibida estará distorsionada o incompleta. En este caso, el dispositivo activo interrumpe la comunicación y reintenta el proceso, generalmente utilizando un esquema de espera aleatoria para evitar nuevas colisiones.

2. Anti-colisión en modo activo y peer-to-peer: en los modos activo y peer-to-peer, donde ambos dispositivos pueden generar sus propios campos electromagnéticos, se utilizan técnicas de acceso por división temporal (Time Division Multiple Access (TDMA)). Aquí, los dispositivos coordinan el tiempo en que cada uno puede transmitir, evitando que las señales se solapen. Si se detecta una colisión, los dispositivos pausarán la transmisión y reintentarán después de un breve intervalo de tiempo aleatorio, reduciendo así la probabilidad de colisiones repetidas.

2.2.4. Mecanismos de seguridad en NFC

Los sistemas NFC incorporan varios mecanismos de seguridad diseñados para proteger la comunicación. Entre ellos se encuentra el cifrado de datos, que asegura que la información transmitida entre dispositivos sea ilegible para terceros no autorizados. Esto se logra mediante la implementación de estándares de cifrado, como el Advanced Encryption Standard (AES), que proporciona una capa adicional de protección contra la interceptación.

La autenticación mutua es otro mecanismo crucial. Antes de que se realice cualquier transferencia de datos, los dispositivos NFC se someten a un proceso de verificación recíproca para confirmar su identidad. Esta autenticación se basa en certificados digitales o secretos compartidos, lo que ayuda a prevenir ataques *man-in-the-middle*.

Para proteger contra los ataques de replay, los sistemas NFC utilizan códigos de un solo uso (One-Time Password (OTP)) y marcas de tiempo. Estos métodos aseguran que las transmisiones capturadas no puedan ser reutilizadas para acceder a servicios sin autorización, añadiendo una capa adicional de seguridad a la comunicación.

Los dispositivos NFC suelen integrar elementos seguros, que son chips especializados diseñados para almacenar información confidencial y realizar operaciones criptográficas de manera segura. Estos elementos aseguran que las claves y otros datos sensibles estén protegidos contra accesos no autorizados y manipulaciones.

Además, se emplean protocolos de inicialización segura que establecen una conexión segura entre los dispositivos al inicio de la comunicación. Este protocolo garantiza que solo los dispositivos que superen una verificación mutua puedan intercambiar datos, protegiendo la integridad del proceso.

Finalmente, en dispositivos multifuncionales, las aplicaciones NFC están aisladas entre sí para evitar que una aplicación comprometida afecte a otras o acceda a información confidencial. Este aislamiento, junto con un control de acceso basado en políticas que regula qué datos pueden ser leídos o escritos por cada dispositivo, asegura que la información se mantenga segura y que los accesos estén restringidos según las reglas establecidas.

Estos mecanismos colaboran para crear un entorno seguro para la comunicación NFC, mitigando los riesgos asociados con la interceptación, manipulación y reutilización de datos, así como con la clonación de dispositivos.

2.2.5. NDEF

NFC Data Exchange Format (NDEF) es un estándar de formato de datos diseñado y utilizado para el almacenamiento y transporte de diferentes tipos de datos entre los multiples y diversos dispositivos NFC, como etiquetas, tarjetas o smartphones. NDEF organiza los datos en mensajes, que a su vez están compuestos por múltiples registros. Cada registro contiene un tipo de carga útil (payload), que puede incluir texto, Uniform Resource Identifier (URI), datos multimedia u otros formatos de datos específicos.

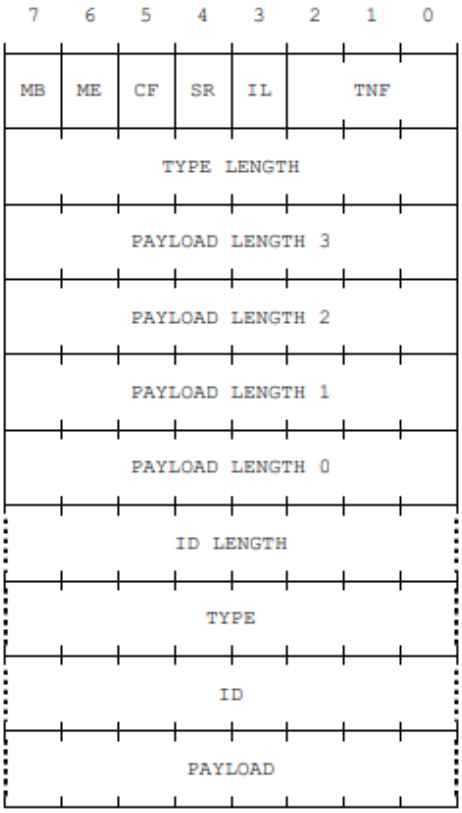

Figura 2.6: Registro NDEF

Un registro NDEF, según la estructura descrita en la 2.6, comienza con un encabezado que identifica el tipo de datos y proporciona información de control adicional. Estos datos son los siguientes:

- **Message Begin (MB)**: Indica si el registro es el primero en el mensaje NDEF. Si está establecido en 1, el registro es el primero.
- **Message End (ME)**: Indica si el registro es el último en el mensaje NDEF. Si está establecido en 1, el registro es el último.
- **Chunk Flag (CF)**: Utilizado para indicar si el registro es una parte de un registro dividido en varios fragmentos.
- **Short Record (SR)**: Indica si el tamaño del campo de longitud es corto (menos de 256 bytes). Si está establecido en 1, el tamaño es corto.
- **Id Length (IL)**: Indica si el registro tiene un identificador de longitud. Si está establecido en 1, hay un campo de longitud de identificador.

- **Type Name Format (TNF)**: Especifica el formato del tipo de registro. Puede ser un tipo bien definido, un tipo específico de aplicación, etc.

Otros datos que aparecen en la estructura del mensaje NDEF son la longitud del payload y su posición dentro de una secuencia de mensajes. Los registros dentro del mensaje pueden ser de diferentes tipos, incluyendo registros bien conocidos (como URLs o números de teléfono), registros Multipurpose Internet Mail Extensions (MIME) para datos multimedia, y registros personalizados definidos por el usuario. Este diseño modular y flexible permite que NDEF se utilice en una amplia gama de aplicaciones, desde pagos móviles hasta la lectura de etiquetas NFC en museos o tiendas.

NDEF es compatible con las principales plataformas móviles, como Android e iOS, y está estandarizado por el NFC Forum [21], lo que asegura la interoperabilidad entre diferentes dispositivos y fabricantes. Gracias a su estructura simple y a la compatibilidad con distintos tipos de datos, NDEF se ha convertido en un elemento fundamental para la implementación de servicios basados en NFC.

2.2.6. Comunicación mediante NFC

En la comunicación NFC, los mensajes se intercambian a través de un proceso estructurado que puede variar según el modo de operación.

En modo activo, la comunicación comienza cuando un dispositivo inicia la transmisión de una señal de RF. El segundo dispositivo detecta esta señal y responde. Ambos dispositivos negocian los parámetros de la comunicación, como la velocidad de transmisión, para asegurar que ambos estén sincronizados. Una vez establecida la conexión, los datos se envían en forma de mensajes encapsulados en el formato NDEF.

En modo pasivo, el proceso es ligeramente diferente. Un dispositivo actúa como lector o escritor y genera una señal de RF que energiza una etiqueta NFC pasiva. Esta etiqueta, al estar pasiva, responde enviando datos modulado en la señal de RF recibida del lector. Los datos se intercambian de manera similar al modo activo, en formato NDEF, permitiendo operaciones de lectura o escritura según lo que se desee realizar.

2.2.6.1. Protocolo de comunicación en NFC

El protocolo de comunicación en NFC se basa en la interacción entre varias capas que gestionan la transmisión de datos:

- **Logical Link Control Protocol (LLCP):**
  - LLCP se encarga de establecer la conexión inicial entre los dispositivos en modo peer-to-peer.
  - Gestiona la negociación de capacidades y el control del flujo de datos.
  - Proporciona corrección de errores y permite la multiplexión, es decir, manejar múltiples servicios a través de una sola conexión.
- **Protocolo de intercambio de datos (NFC Forum Tag Type):**
  - **Type 1 Tag [22]:** Basado en ISO/IEC 14443A, permite operaciones básicas de lectura y escritura con una estructura de memoria sencilla.
  - **Type 2 Tag [23]:** Basado en ISO/IEC 14443A, con una estructura de memoria más flexible y capacidades adicionales como la configuración de áreas específicas de memoria.
  - **Type 3 Tag [24]:** Basado en ISO/IEC 15693, con mayor capacidad de memoria y soporte para operaciones de alta velocidad.
  - **Type 4 Tag [25]:** Basado en ISO/IEC 14443A, ofrece gran flexibilidad en términos de memoria y capacidades, incluyendo la posibilidad de almacenar aplicaciones completas.
- **Protocolo de comunicación en modo pasivo:**
  - **ISO/IEC 14443 [26]:** Utilizado para la comunicación en 13,56 MHz, define el protocolo para tarjetas de proximidad y sistemas de identificación de corto alcance.
  - **ISO/IEC 15693 [27]:** Define un protocolo para etiquetas NFC que operan a distancias mayores, proporcionando una interfaz para la lectura y escritura de datos.

Ademas es importante mencionar tambien el Simple NDEF Exchange Protocol (SNEP) es fundamental para la transferencia de mensajes NDEF entre dispositivos NFC en modo peer-to-peer. SNEP opera sobre el protocolo LLCP, proporcionando un marco para el intercambio eficiente de mensajes NDEF, que pueden variar en tamaño desde simples mensajes hasta archivos más grandes. La función principal de SNEP es definir cómo se envían y reciben estos mensajes, facilitando así la comunicación entre aplicaciones en dispositivos NFC. De esta manera, SNEP complementa a LLCP al permitir que la transferencia de datos se realice de forma estructurada y eficiente.

Capítulo 3

Emulación de un tag NFC

En este capítulo, se aborda la creación de un *applet* en JavaCard diseñado para emular un tag NFC utilizando el estándar NDEF de tipo 4 [25]. Este ejercicio es fundamental para comprender las capacidades de las tarjetas inteligentes JavaCard en la emulación de dispositivos NFC, lo que es esencial para aplicaciones como el intercambio de datos, la autenticación y los pagos sin contacto.

El estándar NDEF tipo 4 es uno de los formatos más utilizados para la comunicación NFC debido a su estructura sencilla y su compatibilidad con una amplia gama de dispositivos. Al implementar un *applet* que emula una etiqueta NFC con este estándar, se exploran conceptos clave como la gestión de archivos en JavaCard, el manejo de comandos APDU y la interacción con lectores NFC.

3.1. Requisitos funcionales

El diseño del *applet* de emulación NFC se fundamenta en la necesidad de cumplir con los estándares especificados por el NFC Forum [21], específicamente con la implementación de una etiqueta de tipo 4. Para ello, es esencial definir los requisitos funcionales del *applet* que permiten la correcta emulación del comportamiento de una etiqueta NFC, asegurando la interoperabilidad con dispositivos lectores NFC conforme a las especificaciones [ISO/IEC 7816-4 [1]] y el formato de intercambio de datos NDEF [28].

Según la especificación [25], el almacenamiento de información en la tarjeta debe organizarse según una estructura de árbol de ficheros. Por tanto, se establece que el applet debe ser capaz de gestionar (crear, seleccionar, leer, escribir, ...) un sencillo sistema de archivos. Debe definir un directorio que contenga al menos los archivos clave definidos, que incluyen el archivo contenedor de capacidades (CC File) y el archivo NDEF, que almacena los mensajes NDEF. Además, se contemplaron las restricciones de acceso, como

la lectura y escritura en el archivo NDEF, dependiendo del estado de la etiqueta (e.g., solo lectura o lectura/escritura).

Finalmente, uno de los requisitos críticos es asegurar que el applet JavaCard sea capaz de manejar versiones del mapa de memoria (Mapping Version), que determinan la estructura de datos y el tamaño máximo del mensaje NDEF que puede ser almacenado. En este sentido, se establece el soporte tanto de la versión 2.0 como de la 3.0 del mapeo, proporcionando compatibilidad con dispositivos antiguos y nuevos.

3.2. Especificación funcional

3.2.1. Estructura de memoria

En una especificación basada en NDEF de tipo 4, el árbol de ficheros generalmente sigue una estructura jerárquica que facilita la organización y el acceso a los datos, dicha estructura se puede observar en la figura 3.1. En la raíz del árbol se encuentra el fichero principal (Card Capability File (CC File)), que actúa como el punto de entrada para la navegación del sistema de archivos. A partir de este fichero raíz, se ramifican múltiples ficheros, cada uno identificado por un tag específico que puede incluir datos NDEF. Los directorios pueden contener ficheros adicionales que almacenan registros NDEF, y cada fichero puede tener una estructura diferente para definir el contenido y el tamaño de los datos. Esta organización permite una navegación eficiente a través de los registros NDEF y facilita la gestión de la información almacenada en el sistema de archivos de la tarjeta.

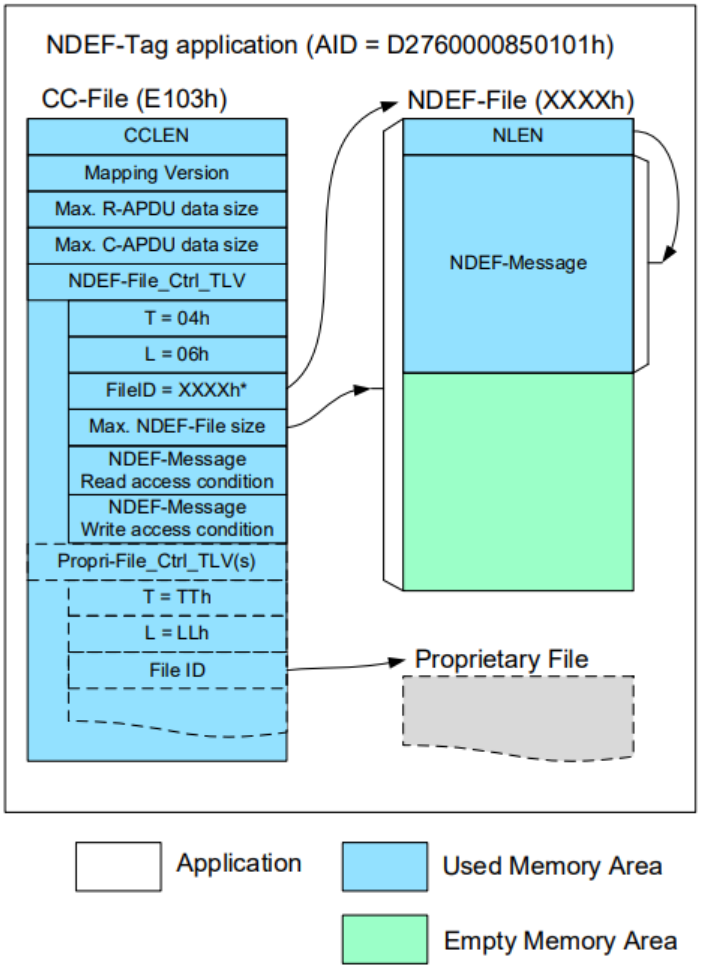

Figura 3.1: Ejemplo de estructura de memoria de una etiqueta de tipo 4

El CC File es un componente crucial en la emulación de una etiqueta NFC de tipo 4. Este archivo contiene metadatos que describen la estructura del sistema de archivos y las capacidades de la etiqueta, permitiendo que los dispositivos lectores comprendan cómo interactuar con el mismo. Su dirección de acceso siempre es la misma teniendo que apuntar a la dirección E103 para poder seleccionar correctamente este fichero.

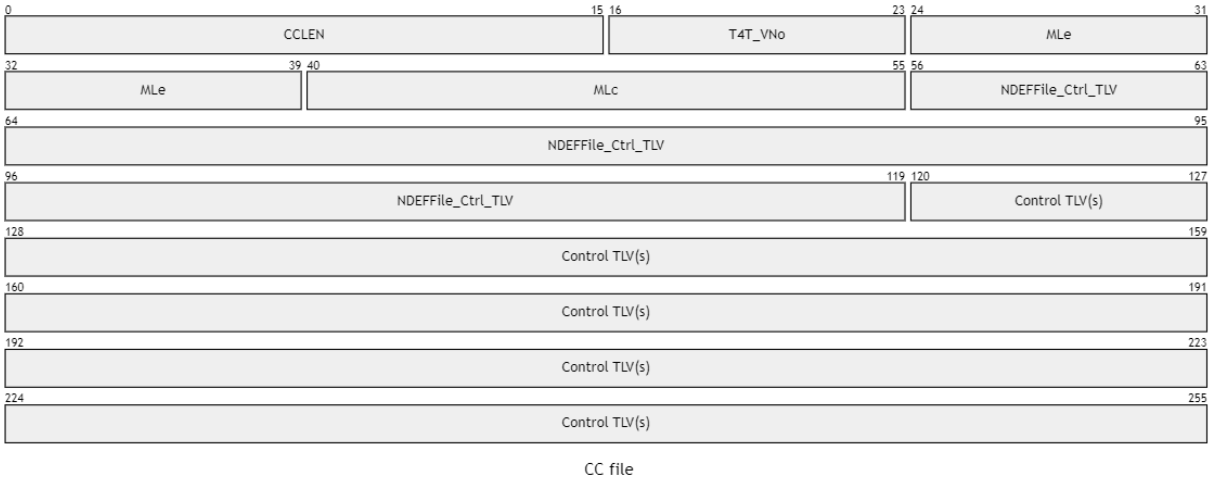

Figura 3.2: Estructura del CC file

La estructura del CC File que se puede apreciar en la 3.2 incluye campos como la longitud del archivo (CCLEN), la versión de la estructura u organización de su contenido (T4T_VNo), y los tamaños máximos de datos que pueden ser leídos (MLe) y escritos (MLc) en una sola operación. Además, el CC File define las condiciones de acceso de lectura y escritura para el archivo NDEF mediante los campos de control TLV (Tag-Length-Value). Estos campos TLV son fundamentales para que el lector entienda qué archivos están disponibles y cómo deben ser gestionados.

Por otro lado, el mensaje NDEF es el formato estándar utilizado para almacenar y transferir datos a través de la tecnología NFC. Un mensaje NDEF se compone de múltiples registros 2.6, donde cada registro contiene un tipo de payload específico, como un texto, un URI, o datos MIME.

En la codificación de mensajes, el formato MIME permite que los dispositivos intercambien datos de diferentes tipos, como texto, imágenes o archivos, especificando el tipo de contenido con un identificador. El formato MIME organiza los tipos de datos en categorías generales como "text"(texto), ïmage"(imagen), .ªpplication"(aplicaciones), y luego especifica un subtipo, como "plain"para texto plano o "png"para imágenes PNG.

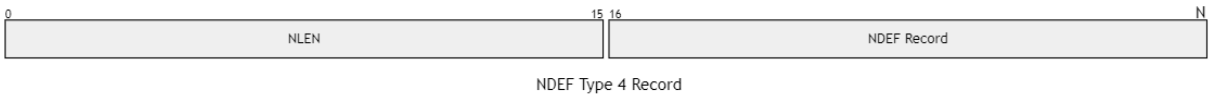

Figura 3.3: Estructura del mensaje NDEF

La estructura del archivo NDEF en el contexto de un Tag de Tipo 4 3.3 incluye dos partes principales: el campo de longitud (NLEN o ENLEN, dependiendo de la versión de mapeo) y el propio mensaje NDEF. NLEN/ENLEN indica el tamaño en bytes del mensaje NDEF almacenado. En la versión de mapeo 2.0, este campo tiene 2 bytes de longitud, mientras que en la versión 3.0, para soportar archivos más grandes, se amplía a 4 bytes. Su dirección de acceso, definida dentro del fichero CC File, se fija para nuestro caso en el identificador de fichero E104.

El mensaje NDEF en sí sigue una estructura descrita en el capitulo anterior. La flexibilidad que aportan estos formatos permiten que las etiquetas NFC puedan utilizarse en una amplia gama de aplicaciones, desde la simple transmisión de URL hasta la compleja transferencia de información estructurada. La correcta implementación y manejo de los mensajes NDEF dentro del *applet* es esencial para asegurar que los datos se intercambien de manera eficiente y segura entre la etiqueta y los dispositivos lectores.

3.2.2. Comandos de operación

Para implementar la emulación de una etiqueta NFC en JavaCard, fue necesario seleccionar y codificar los comandos APDU adecuados que permitieran la comunicación efectiva entre el *applet* y los dispositivos lectores NFC. Basándome en la especificación del NFC Forum para Tags de Tipo 4, se seleccionaron comandos clave como 'SELECT', 'READ_BINARY', y 'WRITE_BINARY', los cuales están definidos en el estándar [ISO/IEC 7816-4].

El comando 'SELECT' fue implementado para permitir la selección de la aplicación NDEF Tag y sus archivos asociados (CC File y Mensaje NDEF). Este comando establece el contexto para posteriores operaciones de lectura y escritura. Para ello, el comando 'SELECT' debía ser capaz de identificar y seleccionar correctamente el identificador de la aplicación (AID) y los identificadores de archivo (File Identifier (FID)) definidos, garantizando que el lector pueda acceder a los datos almacenados en la estructura de archivos del *applet*.

Dicha instrucción dentro de la codificación del *applet* deberá responder a las APDU codificadas con la INS A4. En la tabla 3.1, se puede apreciar la estructura de la APDU completa. Ademas, para nuestro caso omitiremos toda funcionalidad en referencia a la gestión del AID quedándonos unicamente con las funciones de selección dentro del árbol de directorios.

CLA	INS	P1	P2	Lc	Data	Le
00h	A4h	00h	0Ch	02h	Identificador del archivo NDEF o AID	-

Tabla 3.1: Formato del comando APDU de selección

Además, los comandos 'READ_BINARY' y 'WRITE_BINARY' se utilizan para gestionar la lectura y escritura de datos. Estos comandos permiten al lector NFC leer el contenido del archivo NDEF y escribir nuevos mensajes NDEF cuando sea necesario, ademas del resto de ficheros del directorio. Previamente es importante comprender que a nivel de ejecución de comandos hay que realizar una instrucción de 'SELECT' antes de una escritura o lectura para que el *applet* sea capaz de leer o escribir el fichero que previamente se ha seleccionado.

Para realizar correctamente el comando 'READ_BINARY' será necesario procesar las APDU que tengan codificada la INS B0. Ademas en la tabla 3.2 se puede apreciar la estructura completa que tendría una APDU de este tipo.

CLA	INS	P1	P2	Lc	Data	Le
00h	B0h	[Offset]	-	-	Length	-

Tabla 3.2: Formato del comando APDU de lectura

Por último para procesar correctamente las instrucciones de tipo 'WRITE_BINARY' el *applet* deberá reconocer las APDU con INS D6. La estructura completa de la APDU se puede observar en la tabla 3.3

CLA	INS	P1	P2	Lc	Data	Le
00h	D6h	[Offset]	[Offset]	Data	Length Data to be written	-

Tabla 3.3: Formato del comando APDU de escritura

3.3. Implementación del *applet*

Para cumplir las especificaciones descritas, se plantea un *applet* que responda al pseudocódigo descrito en el algoritmo 4.

Para asegurar el funcionamiento adecuado del *applet*, es necesario asignar inicialmente el AID D2:76:00:00:85:01:01. Este identificador garantiza que el lector NFC interprete y gestione los comandos APDU de acuerdo con las especificaciones de una etiqueta de tipo 4, permitiendo la lectura y escritura de datos en formato NDEF. En lugar de simplemente seleccionar un directorio maestro y luego acceder a archivos específicos, como se haría en tarjetas que únicamente soportan la gestión de árboles de ficheros, para el caso de tarjetas JavaCard como las que se emplean en este trabajo, el *applet* se ha diseñado de manera que el AID haga referencia directamente a una aplicación que contiene los

archivos necesarios para la operación. Esto simplifica la interacción, ya que el lector reconoce la aplicación completa como un contenedor de datos, eliminando pasos adicionales de selección de directorio.

Algoritmo 4 Proceso de emulación de una etiqueta de tipo 4

```
1: Recibir APDU
2: Extraer buffer de la APDU
3: Verificar tipo de comando APDU
4: if tipo de comando es SELECT then
5:     if archivo existe then
6:         Procesar selección de archivo
7:     else
8:         Lanzar excepción: Archivo no encontrado
9:     end if
10: else if tipo de comando es READ_BINARY then
11:     if archivo existe Y tamaño solicitado es válido then
12:         Leer datos del archivo seleccionado
13:     else
14:         Lanzar excepción: Lectura no válida
15:     end if
16: else if tipo de comando es WRITE_BINARY then
17:     if archivo existe Y tamaño de los datos es válido then
18:         Escribir datos en el archivo indicado
19:     else
20:         Lanzar excepción: Escritura no válida
21:     end if
22: else
23:     Lanzar excepción: Comando no soportado
24: end if
```

Si se utilizara un AID distinto, debería configurarse que el *applet* se seleccionara por defecto, pudiendo afectar al comportamiento de otras aplicaciones residentes en la tarjeta. En caso contrario, el lector podría no identificar correctamente el *applet* como una etiqueta NFC, lo que afectaría la interoperabilidad en entornos NFC estándar y comprometería la funcionalidad esperada.

Por otro lado, en el proceso de emulación de una etiqueta NFC de tipo 4 utilizando JavaCard, se ha optado por representar la memoria como variables en lugar de ficheros reales, simplificando así la implementación y la gestión de datos. En este enfoque, las estructuras de memoria definidas en la especificación de etiquetas de tipo 4 se codifican

como variables dentro del *applet* de JavaCard 3.4.

```
private static final short FILEID_NONE = (short)0x0000;
private static final short FILEID_NDEF_CAPABILITIES = (short)0xE103;
private static final short FILEID_NDEF_DATA = (short)0xE104;
```

Figura 3.4: Ejemplo de implementación de memoria en el *applet*

Por ejemplo, en lugar de utilizar ficheros físicos para almacenar los datos, se emplean variables como ndefData y apduData (figura 3.5) para mantener los datos NDEF y APDU personalizados, respectivamente. La constante NDEF CAPABILITIES está asociada con la dirección de acceso para el archivo de capacidades NDEF, un ejemplo de esto en la implementación sería lo que observamos en la figura 3.6, mientras que NDEF DATA corresponde al archivo de datos NDEF.

```
private byte[] ndefData = new byte[256]; // Variable para almacenar datos NDEF
private byte[] apduData; // Variable para almacenar datos APDU personalizados
```

Figura 3.5: Ejemplo de implementación de memoria en el *applet*

Las instrucciones APDU como INS_READ_BINARY y INS_WRITE_BINARY se utilizan para leer y escribir datos en estas variables, simulando así el comportamiento de un sistema de archivos sin necesidad de implementar un árbol de ficheros físico. Esta estrategia permite una emulación eficiente del tag, facilitando la manipulación de los datos NDEF y APDU de acuerdo con los requisitos de la especificación NFC sin la complejidad de manejar un sistema de archivos completo.

```
private static final byte[] RESPONSE_CC_FILE = {
    (byte)0x00, (byte)0x0F, // Length of the CC file
    (byte)0x20,             // Mapping Version
    (byte)0x00, (byte)0x3A, // Maximum length of R-APDU
    (byte)0x00, (byte)0x34, // Maximum length of C-APDU
    (byte)0x04,             // T field
    (byte)0x06,             // L field
    (byte)0xE1, (byte)0x04, // File Identifier of the NDEF file
    (byte)0x00, (byte)0xFF, // Maximum NDEF file size
    (byte)0x00,             // Read access without any security
    (byte)0x00              // Write access without any security
};
```

Figura 3.6: Ejemplo de implementación de CC File en el *applet*

3.3.1. Manejo de la comunicación APDU

Los comandos APDU son la única forma de interacción entre el lector NFC y el applet que emula una etiqueta de tipo 4, y su correcta gestión es crucial para asegurar que el sistema funcione según lo esperado. Dicha comunicación (para el caso de la lectura de un fichero) se puede observar en la figura 3.7.

El primer paso en la implementación fue establecer la estructura básica para recibir y procesar los comandos APDU. El *applet* debe sobrescribir el método `process` que recibe como parámetro un objeto APDU. Dentro de este método, se descompone el comando APDU en sus componentes: clase (CLA), instrucción (INS), parámetros (P1 y P2), longitud de datos (Lc), datos (Data) y longitud esperada de la respuesta (Le). Dependiendo de la instrucción recibida, el *applet* ejecuta la operación correspondiente, ya sea selección de archivos, lectura, o escritura.

En la implementación del comando 'SELECT', el *applet* debe verificar si la selección corresponde a la aplicación NDEF Tag o a uno de sus archivos, como el CC File o el mensaje NDEF. Si la selección es correcta, el *applet* configura el entorno adecuado para las siguientes operaciones, como lectura o escritura, dependiendo del archivo seleccionado. En caso de que la selección no sea válida, el *applet* responde con el código de estado 6A82 (Archivo no encontrado), indicando que no se ha podido localizar el archivo requerido.

Además, para los comandos de lectura (READ_BINARY) y escritura (WRITE_BINARY), se implementaron mecanismos para gestionar la transferencia de datos. El *applet* debe manejar las operaciones de acuerdo con los tamaños máximos definidos en el CC File (MLe y MLc), asegurando que no se excedan los límites y que las operaciones sean consistentes con la especificación [ISO/IEC 7816-4]. La implementación incluyó la verificación de la longitud de los datos y el manejo adecuado de fragmentos de datos en casos donde el tamaño de la operación excede los límites permitidos.

Finalmente, la gestión de los códigos de estado (SW1 y SW2) es esencial para una comunicación robusta. El *applet* debe responder con el código 9000 para indicar una operación exitosa, o con códigos de error específicos cuando una operación falla, como 6700 para indicar un error en la longitud de los datos o 6982 para indicar una condición de seguridad insatisfecha. Este manejo detallado asegura que el lector NFC pueda interpretar correctamente los resultados de cada comando y actuar en consecuencia.

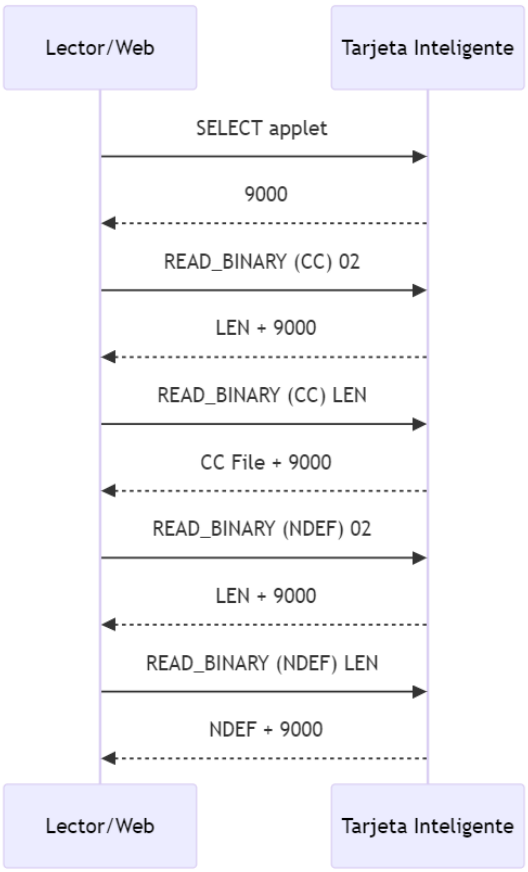

Figura 3.7: Secuencia de lectura de un mensaje NDEF

3.4. Pruebas en dispositivos reales

Para validar la correcta implementación del *applet* de emulación NFC, se llevaron a cabo pruebas exhaustivas en dispositivos reales. Estas pruebas fueron esenciales para asegurar que el *applet* funcionara según lo esperado en un entorno de producción, interactuando con diferentes lectores NFC y dispositivos móviles.

El primer paso en las pruebas fue cargar el *applet* mediante la herramienta Global Platform Pro [29] en una tarjeta JavaCard real utilizando un lector de tarjetas compatible. Una vez cargado, se verificó la correcta inicialización del *applet* y su capacidad para responder a comandos básicos enviados desde un lector NFC estándar. Durante estas pruebas iniciales, se observó que el *applet* respondía adecuadamente a comandos 'SELECT' y 'READ', confirmando que los archivos y la aplicación NDEF podían ser seleccionados correctamente.

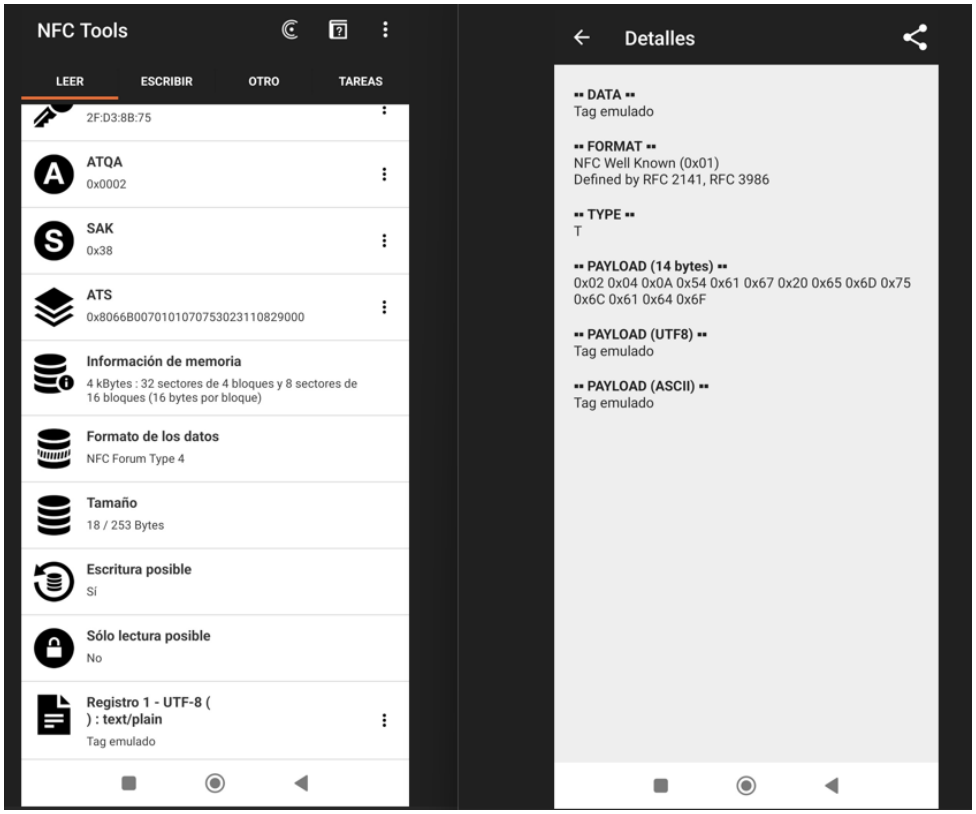

Figura 3.8: Prueba de lectura desde móvil del *applet* de emulación de un tag.

Como se aprecia en la figura 3.8, al leer la tarjeta inteligente, se ha codificado el mensaje predefinido "Tag emulado". Para acceder a el al acercar la tarjeta al lector ha seguido la secuencia indicada en la figura 4.1. El mensaje recibido por tanto es el resultado de leer la etiqueta previamente codificada en el *applet*, la cual vemos en la figura 3.9

```
private static final byte[] RESPONSE_NDEF_FILE_DEMO1 = {
    (byte)0x00, (byte)0x12, // Longitud del archivo NDEF
    (byte)0xD1, (byte)0x01, (byte)0x0E, (byte)0x54, // Cabecera y tipo de mensaje
    (byte)0x02, // Longitud del codigo de lenguaje
    (byte)0x04, (byte)0x0A, // Codigo de lenguaje
    // Payload
    (byte)0x54, (byte)0x61, (byte)0x67, (byte)0x20, (byte)0x65,
    (byte)0x6D, (byte)0x75, (byte)0x6C, (byte)0x61, (byte)0x64,
    (byte)0x6F
};
```

Figura 3.9: Etiqueta por defecto instalada en el *applet*

Posteriormente, se realizaron pruebas más avanzadas enfocadas en la transferencia de datos NDEF entre el *applet* y dispositivos móviles. Utilizando varios modelos de teléfonos con capacidades NFC, se probó la escritura de un mensaje NDEF desde dispositivos móviles. Estas pruebas incluyeron la verificación de la correcta interpretación de los datos, como URL y textos, por parte de los teléfonos, así como la capacidad del *applet* para manejar solicitudes de lectura de gran tamaño. Se identificaron y resolvieron algunos problemas menores relacionados con la fragmentación de datos, mejorando la consistencia de la transmisión de datos.

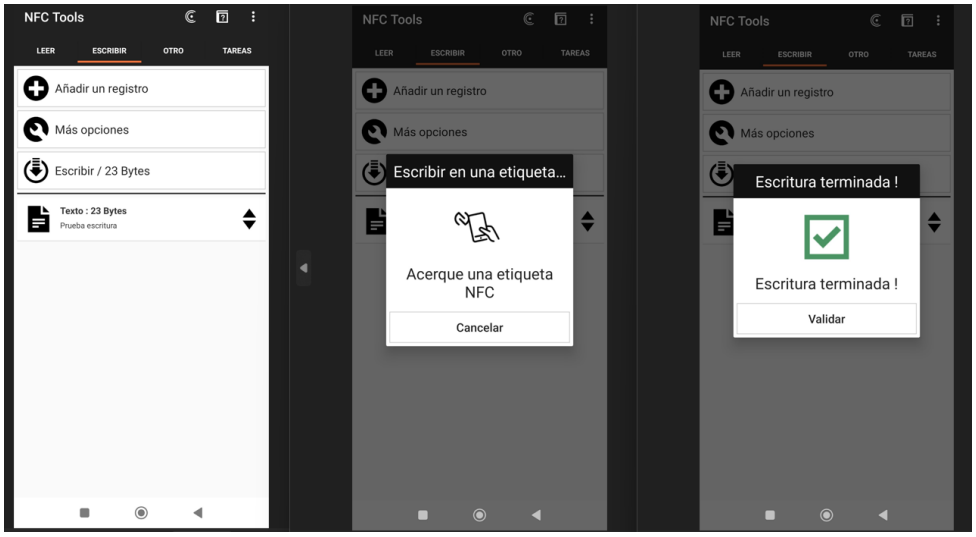

Figura 3.10: Prueba de escritura desde móvil del *applet* de emulación de un tag

En la figura 3.10 se observa la operación de escritura realizada desde un dispositivo móvil, dicha escritura responde a la secuencia de comandos que se observa en el diagrama de la figura 3.11.

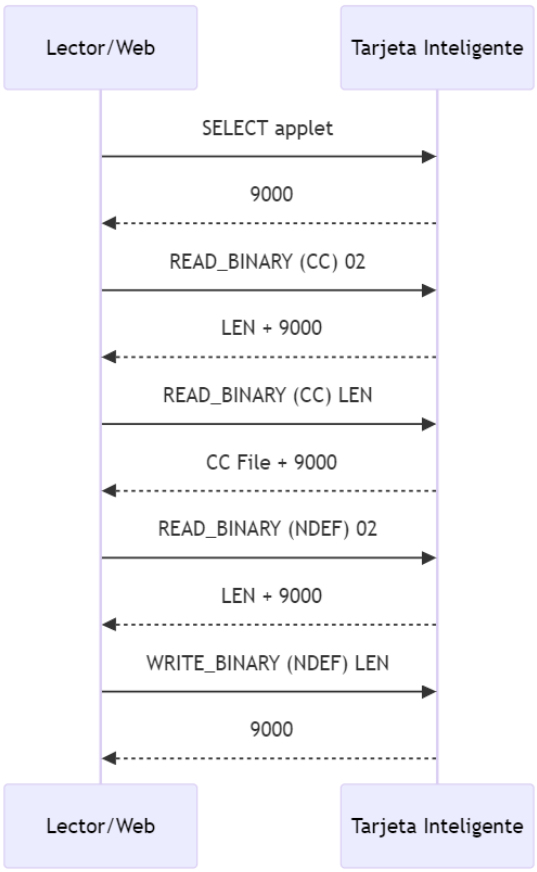

Figura 3.11: Secuencia de mensajes para la escritura de una etiqueta

El mensaje NDEF enviado desde el dispositivo móvil tiene la codificación de un NDEF de tipo texto normal NDEF: 03 0D 54 02 65 73 0B 50 72 75 65 62 61 20 65 73 63 72 69 74 75 72 61. La descripción de cada campo es la siguiente:

- `03` (MB, ME, CF, SR = 1, IL = 0, Type Length = 1)
- `0D` (Payload Length: 13 bytes, incluyendo el lenguaje y el texto)
- `54` (Type: Text)
- `02` (ID Length: 0, Language Code Length: 2 bytes)
- `65 73` (Language Code: 'es')
- `0B` (Length of Text: 11 bytes)
- `50 72 75 65 62 61 20 65 73 63 72 69 74 75 72 61` (Texto: "Prueba escritura")

Posteriormente se procedió a comprobar que realmente se había escrito el contenido en la tarjeta, realizando de nuevo el procedimiento de lectura, dando un correcto resultado como se aprecia en la figura 3.12, ya que el contenido que ahora se lee en la tarjeta es el que previamente habíamos escrito ("Prueba escritura").

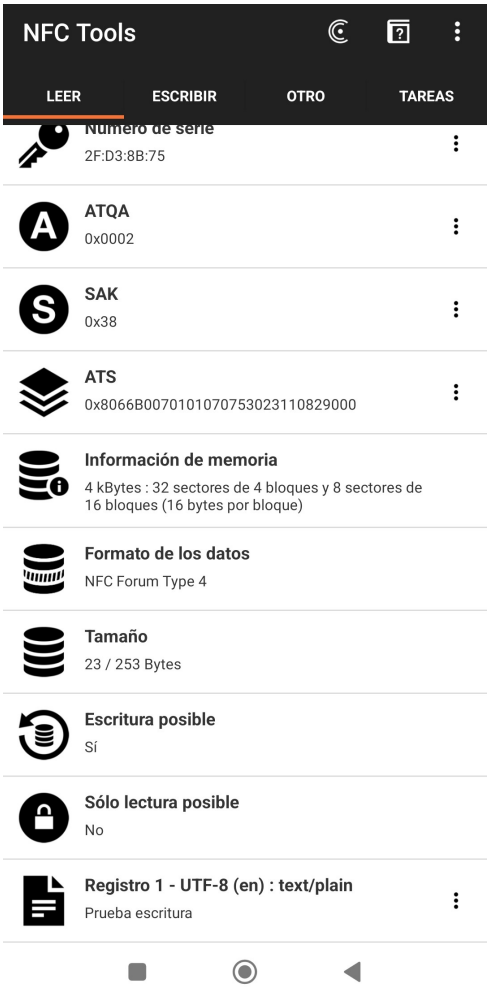

Figura 3.12: Prueba de verificación de escritura del contenido en la etiqueta

Finalmente, se realizaron pruebas en entornos con múltiples lectores NFC para asegurar la interoperabilidad del *applet*. Estos lectores variaban en marca y modelo, lo que permitió verificar que el *applet* funcionara correctamente en una amplia gama de escenarios. Las pruebas incluyeron operaciones de lectura y escritura, así como la respuesta del *applet* a situaciones de error simuladas, como interrupciones en la comunicación y comandos malformados. Los resultados mostraron que el *applet* era robusto y capaz de manejar una amplia variedad de condiciones operativas, lo que confirmó su preparación para el

despliegue en entornos reales.

Estas pruebas en dispositivos reales fueron cruciales para asegurar que el *applet* no solo cumpliera con las especificaciones técnicas, sino que también proporcionara un rendimiento fiable en situaciones prácticas, alineado con las expectativas de los usuarios y los estándares de la industria, para así poder avanzar en la especificación del protocolo de comunicación al que se refiere este proyecto.

Capítulo 4

Protocolo de encapsulación de APDU sobre NDEF

En el capítulo anterior, se ha presentado el proceso de emulación de una etiqueta NDEF tipo 4 utilizando una tarjeta JavaCard. Dicha implementación permite que la tarjeta se comporte como una etiqueta NFC estándar, que puede ser leído por dispositivos compatibles con NFC para intercambiar información estructurada según el formato NDEF. Sin embargo, esta solución, aunque funcional, está limitada a la emulación básica de un tag y no explora en profundidad la personalización ni las capacidades avanzadas de las tarjetas inteligentes para manejar datos de forma más dinámica y segura.

En este capítulo, se abordarán las ventajas de optar por un protocolo de comunicación más sofisticado entre la tarjeta inteligente y los dispositivos lectores. La utilización de mensajes NDEF personalizados para encapsular instrucciones APDU nos permite la utilización de la tarjeta más allá de simplemente emular un tag NFC, es decir ser capaces de mandar APDU que sean ejecutadas en la tarjeta para obtener todas sus capacidades. Esta aproximación no solo amplía las posibilidades de interacción, sino que también integra medidas de seguridad adicionales, como la autenticación y la encriptación, que no son posibles con una simple emulación de una etiqueta NFC.

Es importante aclarar que la necesidad de utilizar este enfoque surge del objetivo de interactuar a través de aplicaciones web que únicamente permiten la interacción empleando mensajes NDEF. En otras palabras, se busca encapsular instrucciones APDU en mensajes NDEF para facilitar la comunicación entre aplicaciones web y tarjetas inteligentes. Esto responde a la limitación de que, mientras que las aplicaciones nativas pueden comunicarse directamente con la tarjeta, las aplicaciones basadas en tecnologías web solo pueden hacerlo a través de mensajes NDEF.

Por lo tanto, esta estrategia no solo ofrece un mayor nivel de funcionalidad, sino que también permite superar las restricciones de comunicación impuestas por las aplicaciones web.

4.1. Descripción del protocolo

Dado que la comunicación con los lectores NFC debe adherirse a unicamente leer y escribir en etiquetas NFC deberemos apoyarnos en esa filosofía para encapsular en ellas de algún tipo las APDU que queremos ejecutar. Para lograrlo, hemos encapsulado la información en un nuevo tipo de registro NDEF, específicamente diseñado para este propósito. Aunque podría haberse implementado en un único tipo de mensaje, se ha optado por dividirlo en dos tipos MIME distintos por razones de diseño. De este modo, `application/apdu-req` se utiliza para codificar las instrucciones APDU, y `application/apdu-res` para decodificar las respuestas, lo que mejora la claridad y la separación de responsabilidades en la comunicación.

4.1.1. Estructura de los mensajes NDEF

Dividir los mensajes en los tipos `application/apdu-req` y `application/apdu-res`, tal como se ha comentado anteriormente, se debe principalmente a un motivo de diseño, el cual responde a la facilidad tanto a la hora de recibir en la tarjeta inteligente como en la web para desencapsular la información del mensaje principal y procesarlo de una manera eficiente.

TNF	Type Length	Payload Length	Tipo MIME	Payload
0xD2	0x14	0xXX	`application/apdu-req`	-

Tabla 4.1: Formato genérico para registro NDEF de tipo `application/apdu-req`

El mensaje `application/apdu-req`, del cual podemos observar su estructura en la tabla 4.1, es utilizado para encapsular la instrucción APDU enviada por el lector NFC a la tarjeta. Dentro del payload será el punto en el que se incluirá la APDU que se quiere procesar. Dentro de este la APDU deberá escribirse en formato `string hexadecimal`, la cual sigue la estructura estándar de cualquier APDU como se ha explicado anteriormente 3.3.

TNF	Type Length	Payload Length	Tipo MIME	Payload
0xD2	0x14	0xXX	`application/apdu-res`	-

Tabla 4.2: Formato genérico para registro NDEF de tipo `application/apdu-res`

El mensaje `application/apdu-res` del cual observamos su estructura en la tabla 4.2, encapsula la respuesta a una instrucción APDU previamente enviada. Este mensaje sigue una estructura similar al `application/apdu-req`, con la diferencia de que el payload contiene la respuesta a la APDU enviada, la cual deberá ser interpretada por el dispositivo.

Además, en este tipo de mensajes devolverán la respuesta del comando incluso aunque se generen excepciones o códigos de error, para ello el *applet* crea un registro NDEF de este tipo en cuyo payload se encapsulará el código de error, permitiendo así que el receptor sea capaz de interpretarlo.

En el caso que el applet no utilice la encapsulación NDEF, para el manejo de APDU que se reciben de manera estándar, se devuelve directamente el código de estado Status Word (SW) que sigue las convenciones de los comandos ISO 7816 [4]. Esto implica que, si ocurre un error durante el procesamiento de una APDU normal, el applet responde con un código SW específico (como 6A82 para un archivo no encontrado o 6985 para condiciones no satisfechas). Este enfoque es el estándar en las aplicaciones de tarjetas inteligentes y no introduce un formato de mensaje adicional como en los APDU personalizados.

4.1.2. Flujo de comunicación

Para implementar la encapsulación propuesta, partimos del *applet* anterior de emulación de etiquetas NFC, en el cual ya se definen las instrucciones para las operaciones de selección, escritura y lectura. Basándonos en esta estructura, se utilizará el comando de escritura para que el usuario realice la encapsulación empleando el tipo MIME previamente establecido. El flujo del algoritmo se describe en la figura 4.1.

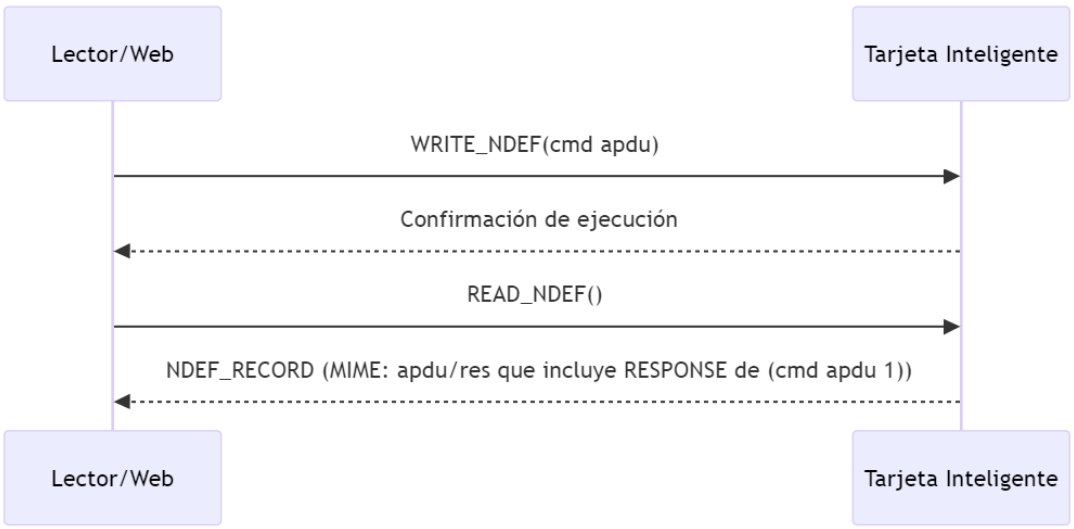

Figura 4.1: Flujo de comunicación

Si hacemos zoom en la instrucción `WRITE_NDEF` siendo rigurosos el flujo de las APDU corresponde al que se observa en la figura 3.11, siendo necesarias 5 instrucciones para poder enviar el mensaje NDEF de tipo `application/apdu-req`. El diagrama de la figura 4.1 se encuentra de manera comprimida el numero de instrucciones necesarias para proceder al envió de nuestra APDU encapsulada. Ademas, la confirmación de ejecución responde no solo a que la escritura se ha realizado correctamente, sino a que se haya ejecutado sin errores la secuencia de 5 instrucciones que conforman el `WRITE_NDEF`.

Del mismo modo si nos fijamos en detalle en el comando `READ_BINARY` el flujo de las APDU corresponde al que se observa en la figura 3.7.

En este punto observamos que para ejecutar una APDU y obtener el resultado de su ejecución, es necesario enviar un mínimo de 10 instrucciones con sus correspondientes respuestas para poder ejecutar correctamente el protocolo. Este flujo introduce una considerable sobrecarga de tráfico, aumentando potencialmente el volumen de datos transmitidos cuando se requiere ejecutar múltiples APDU para interactuar con la tarjeta inteligente.

4.2. Implementación del protocolo

Para entender mínimamente la lógica, en el pseudocódigo incluido en el algoritmo 5 se puede observar el diseño de manera simplificada del funcionamiento del *applet* al recibir una APDU.

Algoritmo 5 Procesado de APDU encapsulada

```
1: Recibir APDU
2: Extraer buffer de la APDU
3: Verificar tipo de comando APDU
4: if tipo de comando es SELECT then
5:     Procesar selección de archivo
6: else if tipo de comando es READ_BINARY then
7:     if processInRead es true then
8:         processInRead = false
9:         Llamar a función ”processCustomApdu”
10:        Leer datos del archivo E104 (con la respuesta de ”processCustomApdu”)
11:    else
12:        Leer datos del archivo seleccionado
13:    end if
14: else if tipo de comando es WRITE_BINARY then
15:    if Type == application/apdu-req then
16:        Guardar payload en bufferNDEF
17:        processInRead = true
18:    end if
19:    Escribir datos (NDEF de otro tipo) en el archivo seleccionado
20: else
21:    Lanzar excepción: Comando no soportado
22: end if
```

Para implementar el algoritmo es necesario crear varias funciones nuevas y modificar las existentes en el *applet* de emulación de la etiqueta NFC. En primer lugar es necesario que dentro de la función de `WRITE_BINARY` seamos capaces de identificar que tipo de mensaje se está escribiendo, para ello verificamos el tipo de mensaje y si este coincide con el tipo MIME `application/apdu-req` guardamos el payload del registro en una variable local de la aplicación llamada 'ndefData', ademas de eso fijaremos una variable llamada `processInRead` a true para indicar que en la siguiente lectura será necesario procesar la APDU encapsulada recibida anteriormente. De esta forma si no coincide con ese tipo procesamos el mensaje recibido de otra manera, en este caso escribiendo de manera directa el mensaje en el fichero E104 para que su posterior lectura pueda ser inmediata.

La operación de ejecución de las APDU encapsuladas se realiza en el proceso de lectura y no en el de escritura, ya que a la hora de implementar el algoritmo observamos que para dotar de aun más atomicidad al applet era más óptimo. De esta manera si algún problema de comunicación surge entre el `WRITE_BINARY` con APDU encapsulada y el `READ_BINARY` con la respuesta, la tarjeta es capaz de mantener o volver al estado anterior sin problemas.

La nueva función de procesado de una APDU encapsulada (`processCustomApdu`) funciona de manera diferente a `process()`(algoritmo 3), ya que al tratarse de una APDU encapsulada no se tienen las herramientas necesarias para comprobar los diferentes bytes recibidos de manera directa, por lo que es necesario realizar esas comprobaciones antes de procesar la instrucción. Una vez verificado que se ha recibido completamente una APDU y que no le falta o sobra información, se procede a su procesamiento, iterando sobre su campo INS. Detectado cual es la instrucción, se llama a la función específica que implementa y procesa la instrucción correspondiente, y que haciendo uso de otra nueva función básica (`generateCustomResponseTag`) se encargará de generar un nuevo mensaje NDEF de tipo `application/apdu-res` en el que se almacenará la respuesta a la instrucción procesada.

Una vez generado este mensaje de respuesta, este se guardará en el fichero `NDEF` (E104) para que el emisor sea capaz de obtener la respuesta.

```
private void processCustomApdu(APDU apdu, byte[] apduNdef, boolean isApdu) {
    if(isApdu == true){
        byte[] buffer = apdu.getBuffer();
        byte ins = buffer[ISO7816.OFFSET_INS];

        switch (ins) {
            case INS_WRITE_APDU_REQ:
                processWriteApdu(apdu);
                break;
            case INS_READ_APDU_REQ:
                processReadApdu(apdu);
                break;
        }
    } else  {
        byte ins = apduNdef[ISO7816.OFFSET_INS];

        switch (ins) {
            case INS_WRITE_APDU_REQ:
                processWriteApduNdef(apduNdef);
                break;
            case INS_READ_APDU_REQ:
                processReadApduNdef(apduNdef);
                break;
        }
    }
}
```

Figura 4.2: Función de procesado de APDU encapsuladas

Otra modificación que realizamos fue en la función de `processCustomAPDU` la cual se puede ver implementada en la figura 4.2. Puesto que se quería soportar tanto la transferencia de APDU a través de la encapsulación en mensajes NDEF, como a través del interfaz nativo ISO 7816-4 o ISO 14443-4, la función debía diferenciar el origen. Con esta mejora ampliamos aún más la compatibilidad del *applet*.

4.3. Verificación en entorno real

Para una primera evaluación del protocolo, se desarrolló un *applet* funcional en el que, a través de dos instrucciones, se pudiera llevar a cabo la lectura (INS 35) y actualización (INS 34) de un texto almacenado. El *applet* soportará tanto el procedimiento nativo de APDU o a través de la encapsulación NDEF, lograndose por ambos métodos el mismo

resultado.

Para las pruebas se utilizaron 2 programas diferentes. Por un lado para la verificación de funcionamiento de las APDU por el envío estándar, se utilizó el programa PcscScriptor [30] el cual nos permite enviar las APDU usando un lector de tarjetas conectado a nuestro ordenador.

Para ello enviaremos la secuencia de instrucciones:

1. Selección: 00 A4 04 00 07 D2 76 00 00 85 01 01
2. Instrucción: 00 34 00 00 04 68 6F 6C 61
3. Selección: 00 A4 04 00 07 D2 76 00 00 85 01 01
4. Instrucción: 00 35 00 00 00

Esta secuencia incluye inicialmente una instrucción para escribir el texto 'hola' y otra instrucción para leer el contenido alojado. Además es necesario seleccionar previamente el *applet* en el que se quiere ejecutar esas instrucciones en la tarjeta, ya que esta tiene varios instalados.

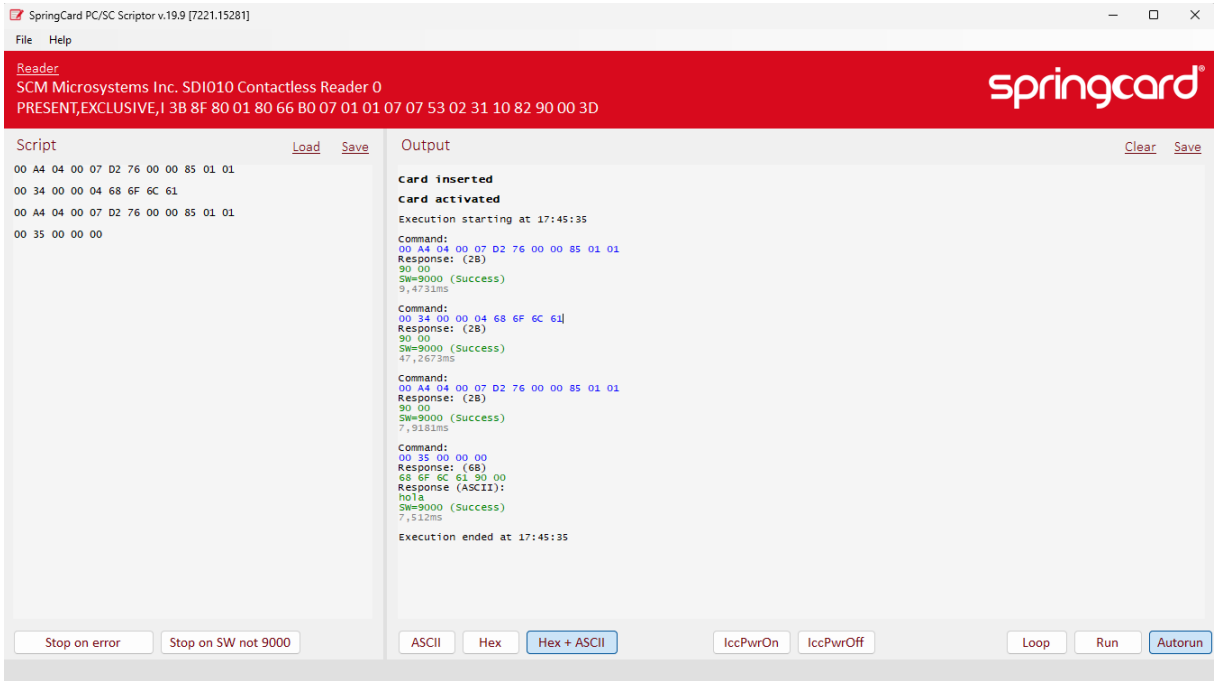

Figura 4.3: Prueba de funcionamiento del algoritmo usando PcscScriptor

Como se puede apreciar en la figura 4.3, el flujo de instrucciones se ejecuta correctamente, lo que demuestra que para APDU mandadas por el método estándar el *applet* funciona correctamente.

Para verificar el uso de mensajes NDEF del protocolo utilizaremos la aplicación móvil NFC Tools [31]. Dicha aplicación nos permitirá escribir mensajes NDEF de cualquier tipo MIME. Mediante esta herramienta veremos de manera secuencial el conjunto de mensajes que son necesarios para procesar una APDU encapsulada.

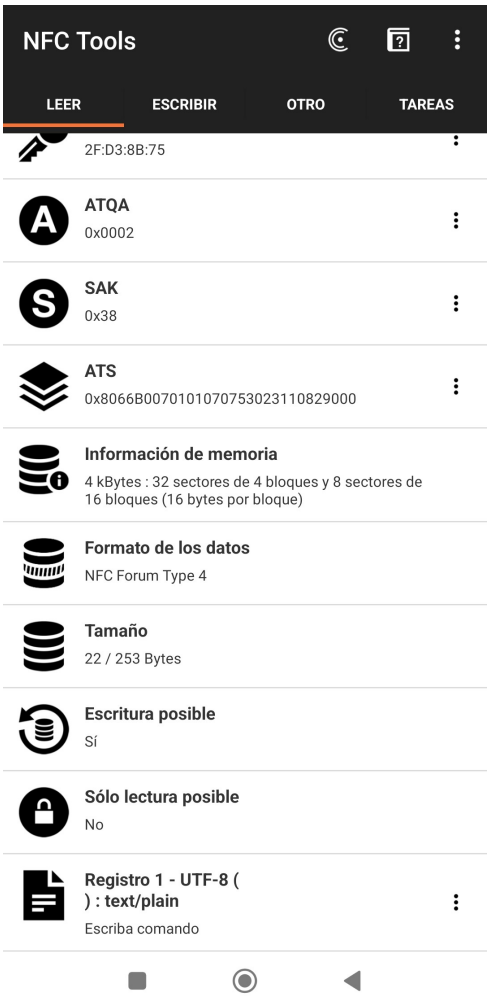

Figura 4.4: Estado inicial del *applet*

En la figura 4.4 se puede apreciar como por defecto si realizamos una lectura de la tarjeta, esta incluye un mensaje de texto genérico en el que se nos indica que está preparada para

recibir mensajes.

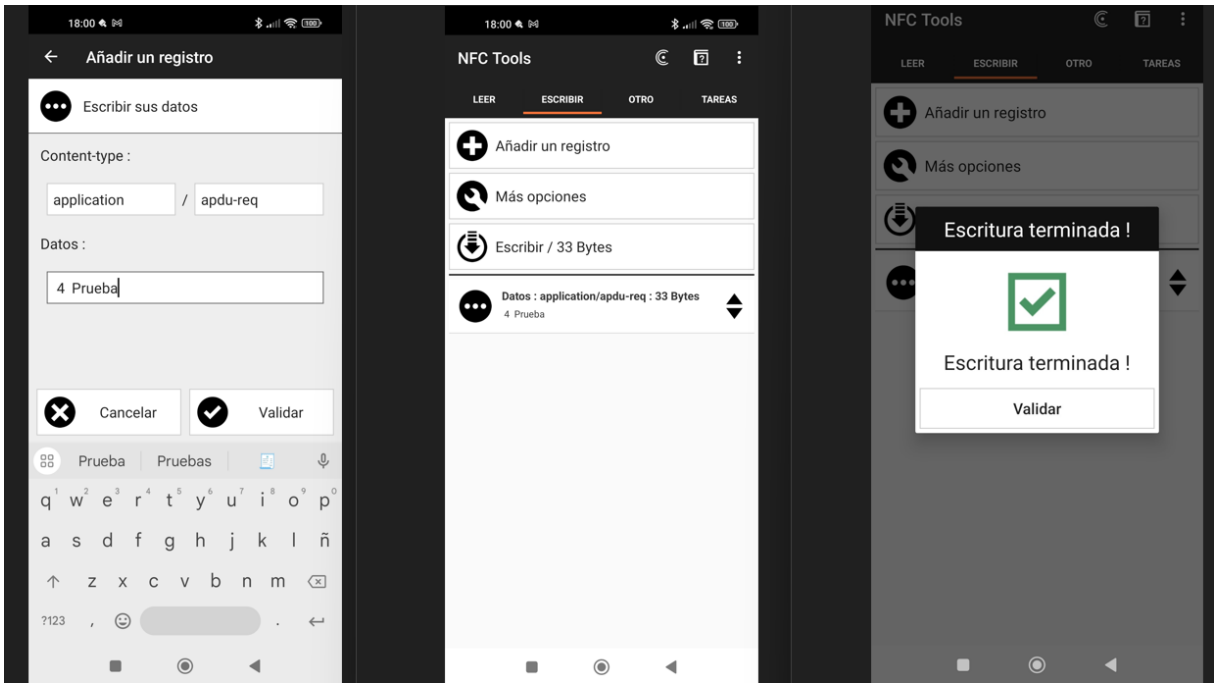

Figura 4.5: Envió de primera APDU con comando usando NDEF

En la figura 4.5 se puede apreciar como realizamos la primera escritura con el comando que queremos ejecutar y su código de éxito de escritura. El mensaje que apreciamos en la imagen es el resultado de escribir el comando 00 34 00 00 06 50 72 75 65 62 61 en formato ASCII (”4 Prueba”).

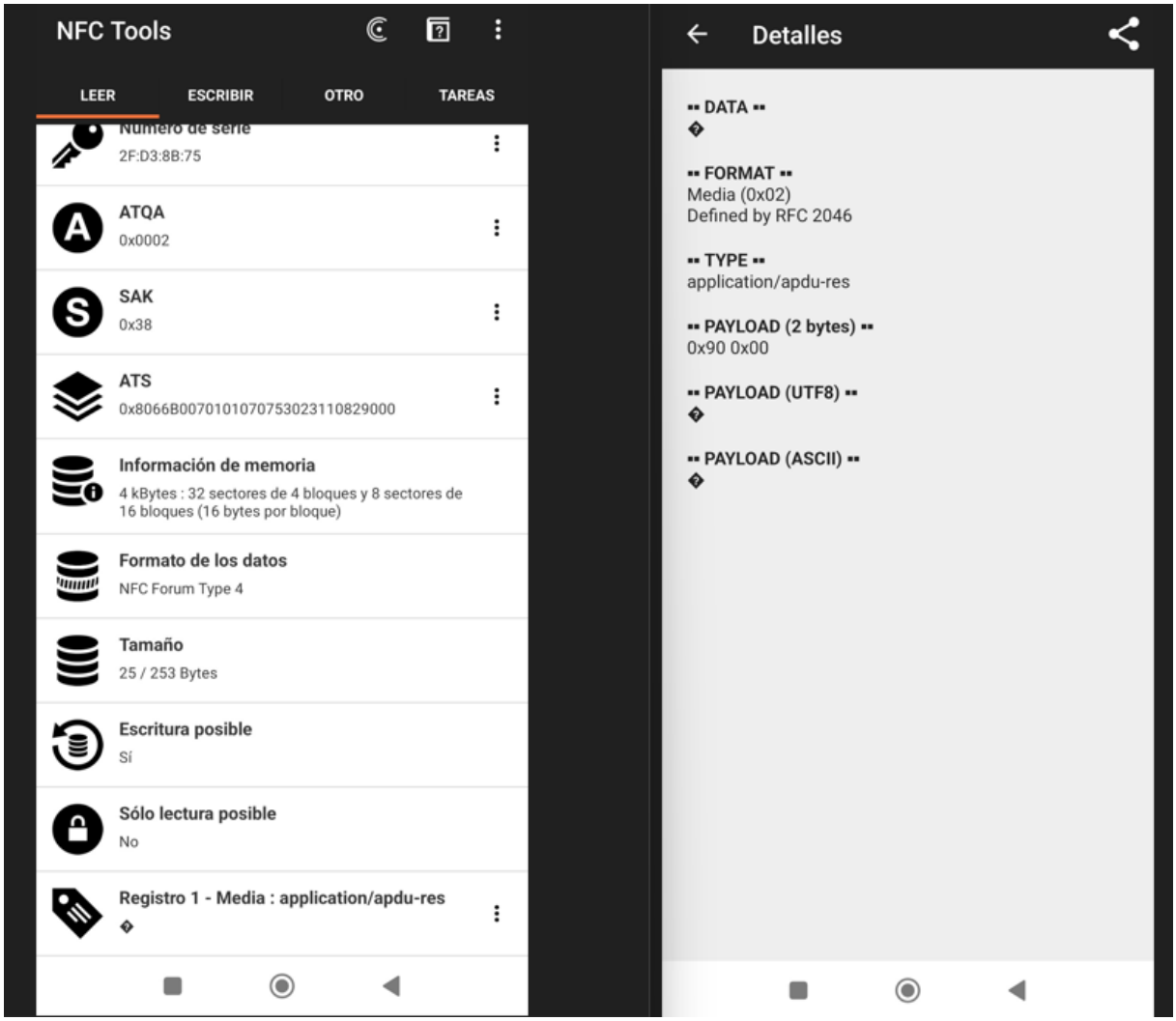

Figura 4.6: Lectura de respuesta de ejecución a la APDU enviada

En la figura 4.6 se puede apreciar como al realizar una lectura de la tarjeta, en este caso, recibimos un NDEF de tipo `application/apdu-res` con el estado de ejecución de la APDU enviada (90 00). Es en este punto, es decir, al realizar esa lectura, cuando realmente el comando se ha ejecutado y generado la etiqueta de respuesta para así mantener la atomicidad en la ejecución.

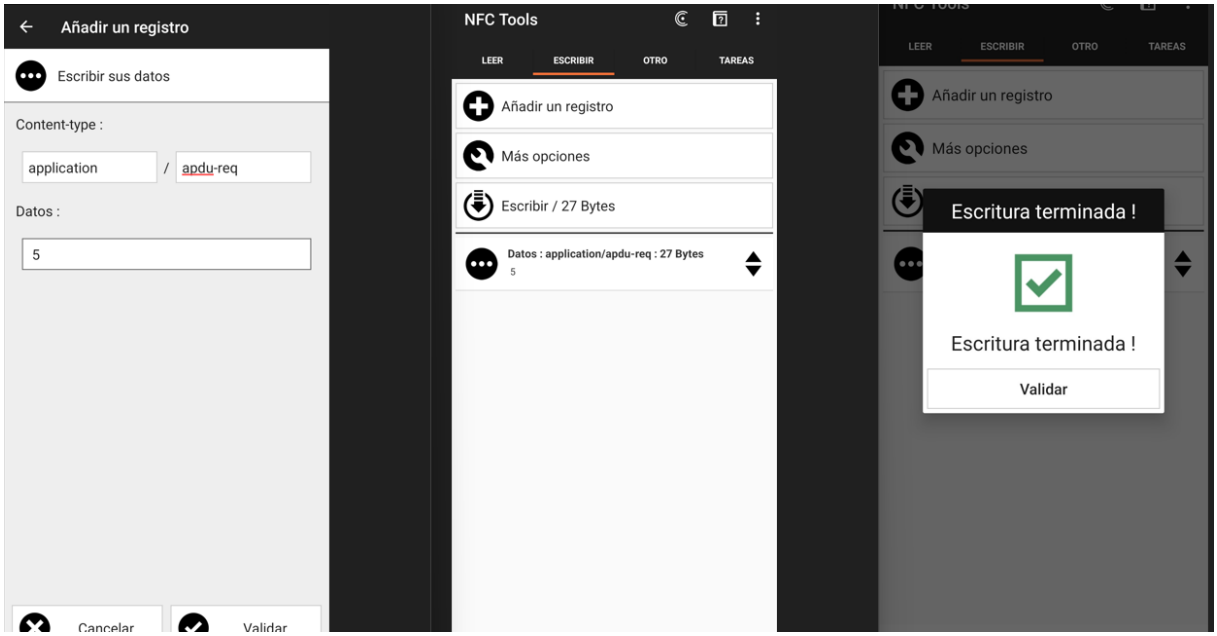

Figura 4.7: Envió de segundo APDU solicitando lectura de APDU anterior usando NDEF

En la figura 4.7 procedemos a enviar la APDU de lectura de la solución, por lo que codificamos la instrucción (00 35 00 00 00), que en la imagen se ve como un 5 y caracteres en blanco (realmente valores hexadecimales que no tienen una representación American Standard Code for Information Interchange (ASCII)) y recibimos la validación de que la escritura se ha realizado correctamente.

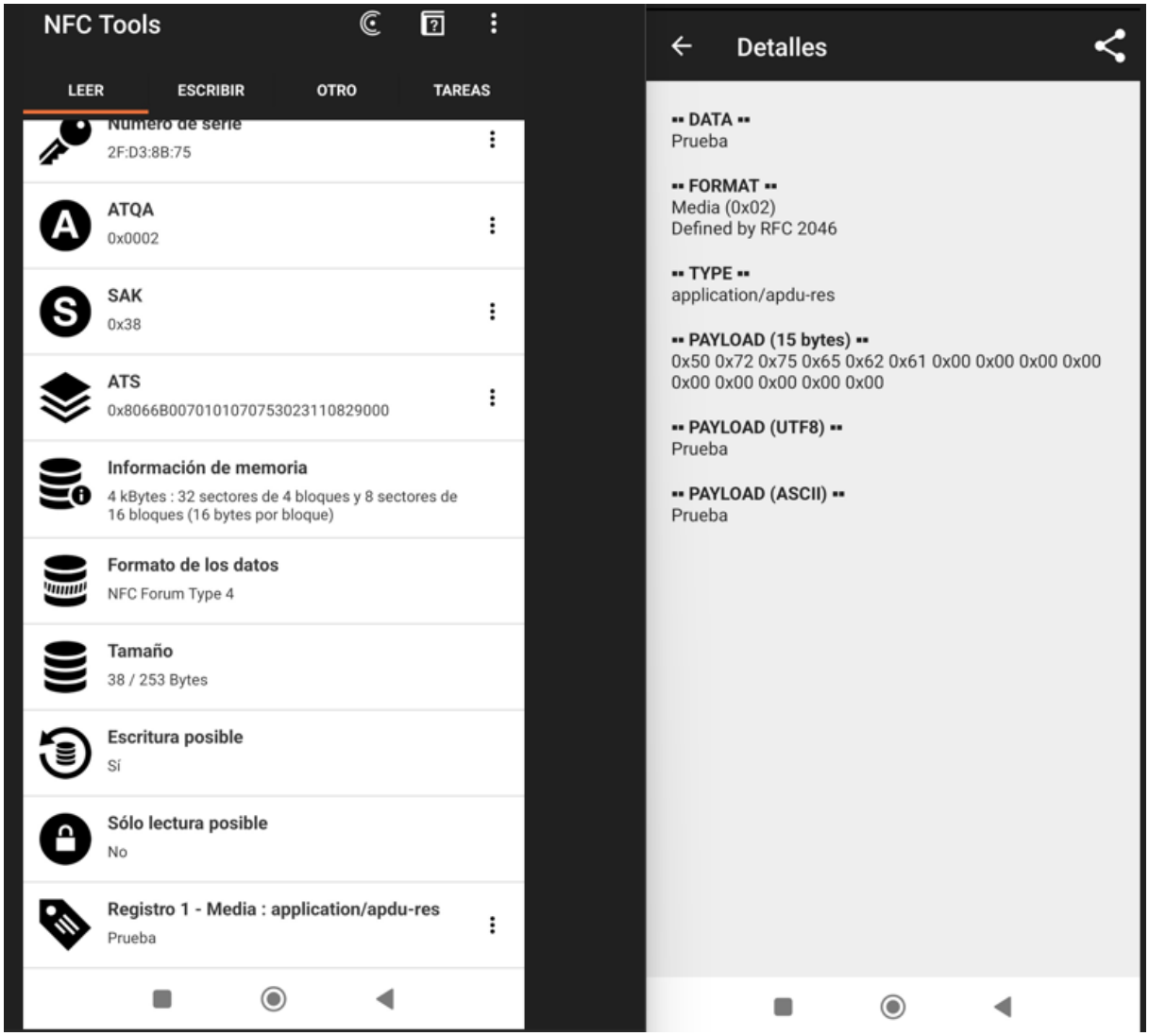

Figura 4.8: Recepción de la respuesta generada por la APDU encapsulada en un mensaje NDEF type `application/apdu-res`

Finalmente en la figura 4.8 observamos como al realizar la lectura de la tarjeta obtenemos como respuesta el texto enviado en la primera APDU encapsulada.

De esta manera se puede verificar que funciona correctamente el *applet* implementado para el caso en el que se este aplicando el algoritmo desarrollado. En la imagen 4.5 se observa como se envía el comando write con su correspondiente confirmación, en la imagen 4.7 se puede apreciar como se envía el comando write en el que se solicita la lectura de la respuesta, mientras que en la imagen 4.8 se puede apreciar la lectura del mensaje NDEF con la respuesta generado, este es de tipo `application/apdu-res` y contiene el mensaje

enviado en la primera APDU del proceso.

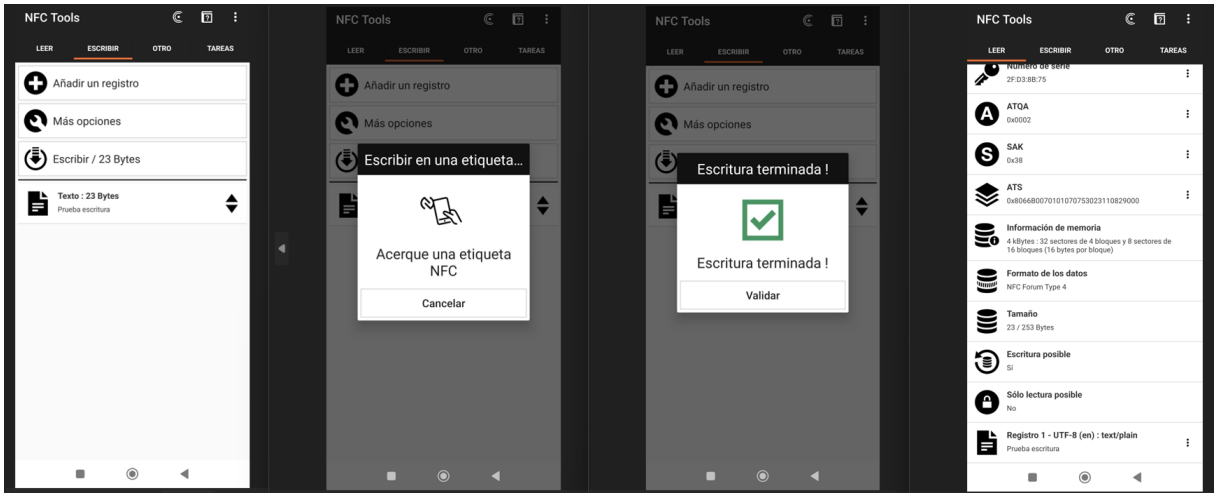

Figura 4.9: Verificación de correcta escritura de otro tipo de mensajes NDEF

A su vez y para comprobar que el *applet* también es capaz de gestionar la escritura y lectura de cualquier otro tipo de etiquetas NDEF, en la figura 4.9 se puede observar como al escribir un NDEF estándar de tipo texto, el *applet* se encarga de guardarlo de manera normal y posteriormente mostrarlo al realizar la lectura.

Capítulo 5

Aplicación web de validación

En el capitulo anterior se presento el diseño e implementación protocolo de comunicación objeto del presente trabajo. En este capitulo se realiza la integración y validación sobre el entorno objetivo, es decir, un página web. Para ello crearemos una web que apoyándose en un conjunto de API será capaz de comunicarse con las etiquetas NFC.

Para validar el correcto funcionamiento del protocolo y demostrar la viabilidad de su uso para aprovechar las capacidades de cómputo de la tarjeta inteligente, yendo más allá del almacenamiento, se planteará una sencilla aplicación web que servirá de interfaz visual de una calculadora implementada como applet en la propia tarjeta.

5.1. Modificaciones al *applet*

Partiendo del *applet* previamente implementado, que ya cuenta con un protocolo de comunicación funcional, se requiere realizar una modificación específica para añadir una nueva función. Esta función debe responder a una APDU con el código de instrucción INS 36.

La nueva funcionalidad se enfoca en realizar una suma de 2 valores introducidos por el usuario, para ello se codifica el dentro del payload de la APDU dos números en formato hexadecimal. En particular, la APDU resultara como se observa en la tabla 5.1, siendo XX el primer numero e YY el segundo.

CLA	INS	P1	P2	Lc	Datos Opcionales
00	36	00	00	02	XX YY

Tabla 5.1: Formato de APDU de comando con valores de ejemplo

El applet debe extraer estos dos bytes del payload y convertirlos a sus valores numéricos correspondientes. Una vez que se tienen estos dos números, el siguiente paso es realizar la suma de ambos valores.

Después de calcular la suma, el *applet* debe generar un mensaje de respuesta que incluya el resultado de la operación en el formato adecuado. Este mensaje de respuesta también debe estar codificado en hexadecimal.

La implementación de esta función en el *applet* implica los siguientes pasos:

1. **Recepción del payload**: El *applet* debe recibir el payload de la APDU y extraer los dos bytes correspondientes a los números.
2. **Conversión de valores**: Convertir los bytes extraídos a valores numéricos. En el contexto de JavaCard, esto generalmente significa convertir los bytes a enteros.
3. **Suma de números**: Realizar la suma de los dos números obtenidos del payload.
4. **Generación del mensaje de respuesta**: Codificar el resultado de la suma en formato hexadecimal y preparar el mensaje de respuesta para su envío.
5. **Envío de la respuesta**: Finalmente, el applet debe enviar el mensaje de respuesta al lector de tarjetas.

Esta extensión permite que el applet no solo responda a comandos anteriores, sino que también realice la operación de suma implementada.

5.2. API `NDEFReader`

La API `NDEFReader` [32] permite a las aplicaciones web interactuar con etiquetas y dispositivos NFC. Esta API es muy útil en aplicaciones donde se necesita leer o escribir información en etiquetas NFC.

Proporciona una interfaz sencilla para acceder a etiquetas NFC compatibles con el estándar NDEF. Para utilizarla, se crea un objeto `NDEFReader`, que permite escuchar eventos NFC y leer datos de estas etiquetas haciendo uso de la interfaz NFC del dispositivo.

Los métodos principales de la API son:

- `scan()`: Inicia la búsqueda de dispositivos NFC cercanos y espera la detección de etiquetas compatibles. Es el método encargado de activar la escucha de eventos relacionados con el escaneo de etiquetas.

- `write()`: Escribe datos en una etiqueta NFC. Los datos deben estar en formato NDEF (NFC Data Exchange Format), asegurando la compatibilidad con dispositivos que utilicen este estándar.

La API también maneja eventos como:

- `reading`: Se activa cuando se detecta y lee una etiqueta NFC, permitiendo acceder al contenido almacenado en ella.
- `readingerror`: Se activa cuando ocurre un error durante el proceso de lectura, por ejemplo, si la etiqueta no es compatible o si la lectura falla.

5.2.1. Compatibilidad

La compatibilidad de esta API con los navegadores se puede observar en la figura 5.1. Esta varía significativamente entre diferentes navegadores. En general, Google Chrome para Android ofrece un soporte robusto para esta API, permitiendo la lectura y escritura de mensajes NDEF siempre que el dispositivo tenga habilitada la funcionalidad NFC. Sin embargo, otros navegadores como Mozilla Firefox y Microsoft Edge no han implementado soporte para NDEFReader, lo que limita la capacidad de utilizar estas funcionalidades en dichos entornos. Además, Safari, particularmente en dispositivos iOS, tampoco ofrece soporte para esta API en la actualidad. Por lo tanto, aunque la API NDEFReader tiene un respaldo sólido en algunos navegadores, su disponibilidad aún está restringida en otros, lo que requiere que los desarrolladores consideren la compatibilidad del navegador al diseñar sus aplicaciones NFC.

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
NDEFReader	No	No	No	No	No	89	No	63	No	15.0	89
NDEFReader() constructor	No	No	No	No	No	89	No	63	No	15.0	89
makeReadOnly	No	No	No	No	No	100	No	69	No	19.0	100
reading event	No	No	No	No	No	89	No	63	No	15.0	89
readingerror event	No	No	No	No	No	89	No	63	No	15.0	89
scan	No	No	No	No	No	89	No	63	No	15.0	89
Secure context required	No	No	No	No	No	89	No	63	No	15.0	89
write	No	No	No	No	No	89	No	63	No	15.0	89

Figura 5.1: Compatibilidad de la API `NDEFReader`

5.3. Desarrollo web

Para el desarrollo de la web se programó directamente en HyperText Markup Language (HTML) utilizando Cascading Style Sheets (CSS) para dar estilo y los propios `javascripts` para dar funcionalidad a la misma. La estructura básica se puede ver en la figura 5.2. Como se puede apreciar tiene una estructura bastante básica permitiendo introducir dos valores numéricos además de 2 botones, uno que realizará la operación de escritura y otro la operación de lectura con la tarjeta inteligente. Además de eso se añade un apartado a modo de consola de depuración en el que se irán imprimiendo paso a paso las diferentes operaciones.

El uso de 2 botones uno para leer y otro para escribir se realiza con un fin didáctico ya que de esta manera se puede ver la ejecución del protocolo de una manera mas secuencial.

```
<div class="container">
    <h1>Web NFC Sum</h1>

    <!-- Input fields for the numbers -->
    <label for="number1">Number 1:</label>
    <input type="number" id="number1" placeholder="Enter first number">

    <label for="number2">Number 2:</label>
    <input type="number" id="number2" placeholder="Enter second number">

    <button id="writeNDEF">Write NDEF Message</button>
    <button id="readNDEF">Read NDEF Message</button>

    <pre id="response"></pre>

    <!-- Debugging console -->
    <h2>Console</h2>
    <div id="console"></div>
</div>
```

Figura 5.2: Estructura de la web implementada

5.3.1. Integración de la API `NDEFReader`

Para interactuar con etiquetas NFC en aplicaciones web, la API Web NFC [32] ofrece el objeto `NDEFReader`, que proporciona métodos clave para la lectura y escritura de mensajes NDEF. En particular, utilizaremos los métodos `write()` y `scan()`:

- **Método** `write()`: Este método se utiliza para escribir datos en una etiqueta NFC. En la aplicación, se emplea el método `write()` para almacenar un mensaje NDEF en la etiqueta. El mensaje NDEF contiene un comando APDU que representa una operación específica que se desea realizar en la tarjeta inteligente. La función `write()` acepta un objeto que describe el mensaje NDEF a escribir, que en este caso incluye un tipo de registro MIME (`application/apdu-req`) y los datos del comando APDU en formato `ArrayBuffer`. Cuando se ejecuta `write()`, la aplicación se asegura de que el mensaje se transfiera correctamente a la etiqueta NFC, permitiendo que la etiqueta almacene esta información para futuras lecturas.

- **Método** `scan()`: Este método inicia un proceso de escaneo para detectar etiquetas NFC cercanas y leer los mensajes NDEF almacenados en ellas. En la aplicación, se utiliza `scan()` para activar la capacidad de lectura del dispositivo NFC. Cuando se detecta una etiqueta, el evento `onreading` se activa, permitiendo que la aplicación procese los mensajes NDEF leídos. Estos mensajes se muestran en la consola, proporcionando información sobre el contenido de la etiqueta. Si ocurre un error durante el escaneo, la función `onreadingerror` se encarga de manejarlo, notificando al usuario sobre cualquier problema en la lectura de los datos de la etiqueta.

```
const number1 = parseInt(document.getElementById('number1').value, 10);
const number2 = parseInt(document.getElementById('number2').value, 10);

if (isNaN(number1) || isNaN(number2)) {
    logToConsole('Invalid input for numbers.');
    return;
}

const number1Hex = number1.toString(16).toUpperCase().padStart(2, '0');
const number2Hex = number2.toString(16).toUpperCase().padStart(2, '0');
const apdu = `00 36 00 00 02 ${number1Hex} ${number2Hex}`;
const dataBuffer = hexStringToArrayBuffer(apdu);
```

Figura 5.3: Método de codificación de una APDU en la web

Para codificar la APDU que se desea enviar se realizan las instrucciones que se observan en la figura 5.3. Primero se obtienen los valores de los campos de entrada con los identificadores `number1` y `number2` del HTML. Estos se convierten a enteros decimales usando `parseInt`. Si alguno no es un número válido, se registra un error con `logToConsole` y se detiene la ejecución con `return`.

Luego, los números se convierten a formato hexadecimal con `toString(16)`, se ponen en mayúsculas con `toUpperCase()` y se ajustan a dos caracteres con `padStart(2, '0')`.

Con los valores generados, se construye el mensaje APDU, que incluye el código de instrucción (36) y los dos números hexadecimales. El mensaje resultante se convierte en un `ArrayBuffer` mediante `hexStringToArrayBuffer`, listo para su envío a través de la API Web NFC.

5.4. Pruebas y validaciones

Una vez creada la web e instalado el *applet* con la nueva función se procede a realizar las pruebas.

En la figura 5.4 se puede observar como la web inicialmente solicita al usuario los permisos para poder utilizar la interfaz NFC en el dispositivo móvil, para posteriormente mostrar la interfaz en la que el usuario debe introducir los valores a calcular.

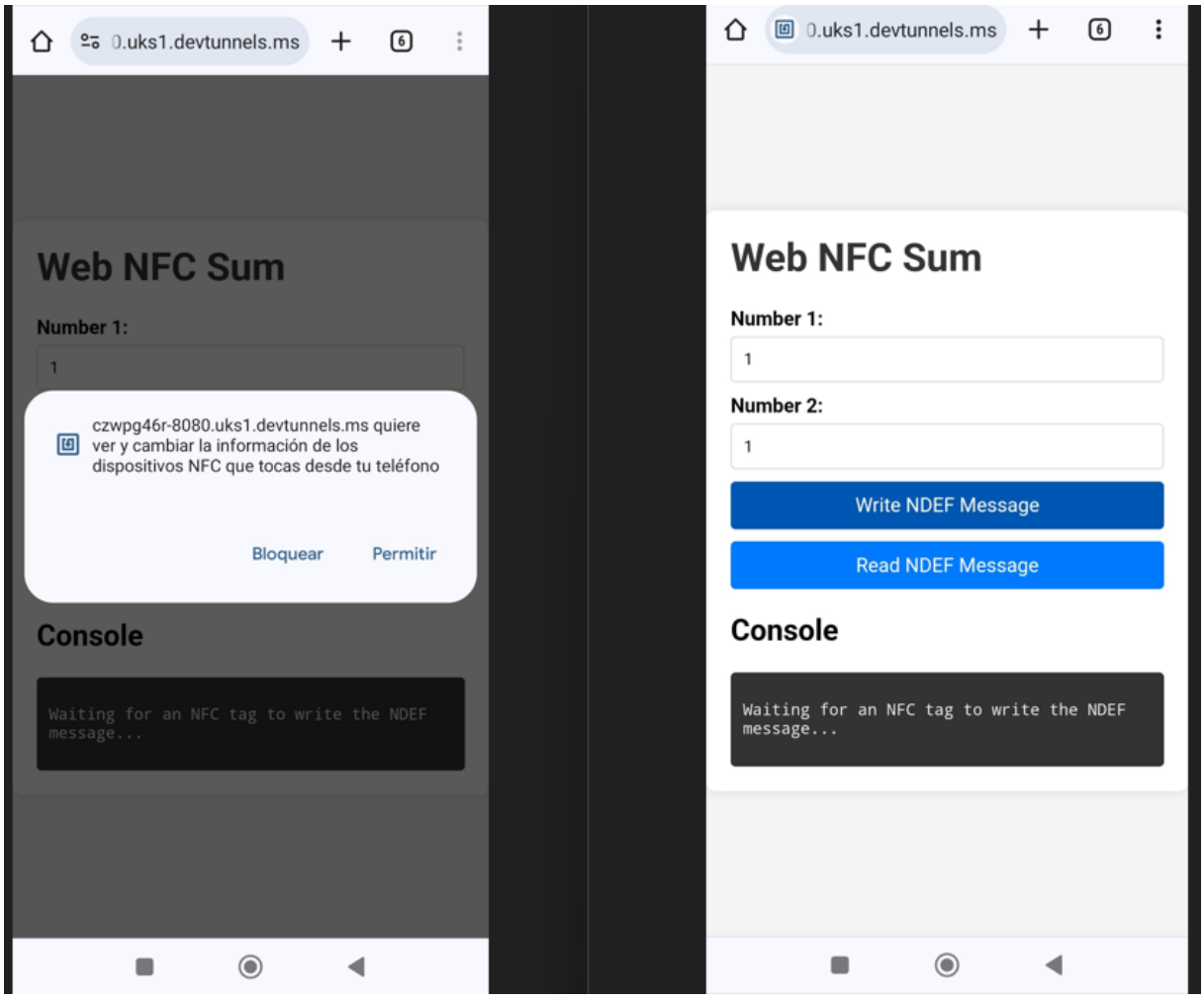

Figura 5.4: Acceso a la web por primera vez

Una vez introducidos los valores se procede a realizar la escritura (figura 5.5), al realizarse acercando la tarjeta inteligente al dispositivo responde con el mensaje de confirmación de que la escritura ha realizado todas las instrucciones correctamente, así como el contenido

que ha añadido al payload del mensaje (en nuestro caso una APDU).

Figura 5.5: Confirmación de escritura en la web

Posteriormente y siguiendo la secuencia del algoritmo (figura 4.1) se procede a pulsar en el botón de lectura para obtener así el resultado de la suma. En la figura 5.6 se observa como en la consola se devuelve el valor de la suma como una etiqueta de tipo `application/apdu-res` con dentro de su payload el resultado de la suma.

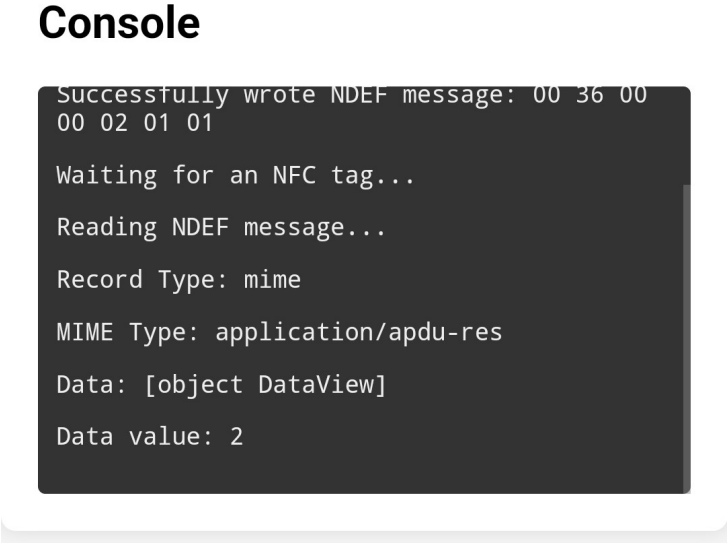

Figura 5.6: Lectura desde la web

Además en la figura 5.7 podemos observar mas detalladamente el contenido de la etiqueta de resultado, dicho análisis ha sido realizado desde la aplicación NFC Reader.

Figura 5.7: Etiqueta de resultado desde NFC Reader

Por último y para verificar la dualidad del *applet* en funcionamiento, se prueba desde un lector conectado al ordenador a enviar la secuencia de APDU:

1. Selección: 00 A4 04 00 07 D2 76 00 00 85 01 01
2. Instrucción: 00 36 00 00 02 01 02

Dicha secuencia realiza la operación de suma directamente con instrucciones APDU nativas sin hacer uso del protocolo de encapsulación NDEF desarrollado. En la figura 5.8 se puede verificar que la operación se ha realizado correctamente ya que se han sumado los numero 1 y 2 y el resultado que se obtiene es 3.

Figura 5.8: Resultado de ejecución de la suma desde un ordenador usando PcscScriptor

Capítulo 6

Conclusiones

En este proyecto se ha desarrollado un protocolo de comunicación para tarjetas inteligentes basado en tecnologías sin contacto (NFC) y NDEF, el cual permite una interacción de bajo nivel a través de APDU entre aplicaciones web móviles y las tarjetas, superando las limitaciones impuestas por los propios sistemas operativos de los dispositivos móviles. Para validar la propuesta, se ha implementado un prototipo funcional que demuestra la capacidad de comunicación entre los dispositivos mediante la lectura y escritura de datos en tiempo real.

6.1. Análisis de los objetivos alcanzados

A lo largo del proyecto, se han logrado alcanzar los principales objetivos establecidos. La exploración de las funcionalidades nativas de los dispositivos móviles, la definición y desarrollo de un protocolo de comunicación basado en NDEF, y la integración de este protocolo con aplicaciones web han sido exitosos. La primera fase del proyecto involucró la comprensión detallada del funcionamiento de las tarjetas inteligentes y su interacción con los dispositivos móviles mediante NFC. Este proceso inicial nos permitió identificar las limitaciones inherentes tanto de las tarjetas como de las API web disponibles.

Tras entender el contexto y las restricciones de las tarjetas JavaCard, así como las capacidades y limitaciones de las interfaces web, se procedió a diseñar un protocolo adaptado a estas condiciones. La definición del protocolo se basó en la experiencia adquirida y en la necesidad de superar las restricciones de comunicación tradicionales. La implementación de este protocolo se realizó con el objetivo de facilitar una comunicación fluida y eficiente entre las tarjetas inteligentes y los dispositivos móviles, sin depender de sistemas propietarios ni aplicaciones nativas. Esto permitió el acceso a las funcionalidades avanzadas de las tarjetas directamente desde el navegador web.

Durante el desarrollo, se abordaron y solucionaron diversos problemas técnicos relacionados con la implementación del protocolo y la integración del applet de emulación de etiquetas NFC. Se realizaron ajustes y optimizaciones para mejorar el rendimiento del sistema, comparando el comportamiento entre la implementación nativa y el enfoque basado en NDEF. Las pruebas iniciales y posteriores validaciones del sistema en distintos dispositivos móviles demostraron que la solución propuesta no solo cumplía con los requisitos funcionales, sino que también operaba de manera segura y eficiente en entornos reales.

El protocolo fue sometido a rigurosas pruebas para evaluar su correcto funcionamiento en términos de seguridad y rendimiento. Estas pruebas incluyeron la verificación de la interacción entre las tarjetas y los dispositivos móviles, así como la eficiencia en el intercambio de información. La seguridad del sistema fue evaluada positivamente, garantizando el flujo de datos en la comunicación.

6.2. Mejoras y propuestas futuras

Aunque el proyecto ha cumplido con los objetivos planteados, existen diversas áreas donde se pueden implementar mejoras, con prioridad en la optimización del manejo de las APDU encapsuladas. Actualmente, el envío de comandos APDU mediante el protocolo NDEF requiere un número elevado de instrucciones debido a la encapsulación de cada APDU individual en un mensaje NDEF separado. Se podría investigar la posibilidad de reducir este número optimizando la codificación de varias APDU en un único mensaje `apdu-req`. Esto permitiría una mayor eficiencia en el intercambio de datos, reduciendo la sobrecarga y mejorando el rendimiento del sistema, especialmente en escenarios de comunicación intensiva.

Un enfoque similar al de la codificación de múltiples APDU en un solo mensaje se ha utilizado en el contexto de los mensajes SMS en el protocolo GSM [33]. En GSM, es posible enviar varios Short Message Service (SMS) concatenados utilizando un método que encapsula múltiples mensajes en una única secuencia mediante técnicas Type-Length-Value (TLV). Esta técnica permite enviar un conjunto de mensajes en una sola transmisión, reduciendo así la sobrecarga y mejorando la eficiencia en la comunicación. Aplicar una estrategia similar en la encapsulación de las APDU podría proporcionar beneficios similares en términos de reducción de la cantidad de mensajes y mejora del rendimiento general del sistema.

Otra línea de investigación prioritaria sería permitir el uso de elementos compartidos (*shareable objects*) entre diferentes applets de la tarjeta inteligente. Sin embargo, lo que realmente buscamos es desarrollar un applet único para la gestión del protocolo que pue-

da interactuar de manera eficiente con otros applets ya presentes en la tarjeta. En lugar de enfocarnos en el uso de datos compartidos, la prioridad sería encontrar maneras de interactuar que faciliten la integración del protocolo de forma sencilla. Por ejemplo, al compartir ciertos datos o funcionalidades clave entre applets, se podrían implementar casos de uso más complejos, como autenticaciones cruzadas o la reutilización de claves de seguridad, sin la necesidad de duplicar información o procesos.

Adicionalmente, se pueden explorar otras mejoras de rendimiento mediante una mayor integración con tecnologías emergentes, como el IoT y la realidad aumentada, lo que abriría nuevas oportunidades en cuanto a la interacción con el entorno. La interoperabilidad del protocolo también podría ampliarse para soportar un mayor número de plataformas y dispositivos, asegurando así la adopción universal de la solución en diferentes ecosistemas tecnológicos.

También es importante considerar la interacción segura en el intercambio NDEF, implementando datos cifrados y estableciendo sesiones seguras entre el móvil y la tarjeta inteligente. Esto garantizará la protección de la información durante la transmisión y mejorará la seguridad general del sistema. Además, se recomienda analizar las nuevas versiones de JavaCard y las tarjetas inteligentes disponibles, para plantear mecanismos o restricciones de uso que habiliten el applet únicamente en casos de necesidad y bajo ciertas condiciones, por ejemplo limitar el applet a que solo se pueda utilizar mediante la interfaz inalámbrica de la tarjeta. Esto permitirá una mayor seguridad y un uso más controlado del applet, adaptándose a las capacidades y restricciones de las tecnologías emergentes.

Bibliografía

[1] Iso/iec 7816-4: Identification cards - integrated circuit cards - part 4: Organization, security and commands for interchange, 2005. Establece los comandos APDU para la comunicación con tarjetas inteligentes.

[2] Iso/iec 18000: Information technology - radio frequency identification for item management, 2015. Estándar para la tecnología de identificación por radiofrecuencia (RFID) que cubre las especificaciones para distintos tipos de sistemas RFID.

[3] Iso/iec 7810: Identification cards - physical characteristics, 2019. Establece las características físicas de las tarjetas de identificación, incluyendo tamaño y formato.

[4] Iso/iec 7816: Identification cards - integrated circuit cards, 2020. Conjunto de normas que especifica los requisitos para tarjetas inteligentes con circuito integrado, incluyendo la organización, seguridad y comandos de intercambio.

[5] Iso/iec 7816-3: Identification cards - integrated circuit cards - part 3: Cards with contacts - electrical interface and transmission protocols, 2006. Especifica el interfaz eléctrico y los protocolos de transmisión para tarjetas inteligentes con contactos.

[6] Java card platform specification, 2021. Especificación de la plataforma Java Card, que define el entorno y las API para desarrollar aplicaciones en tarjetas inteligentes con tecnología Java.

[7] Multos: Multi-application operating system for smart cards, 2024. URL `https://www.multos.com/`. Descripción del sistema operativo MULTOS para tarjetas inteligentes.

[8] Cardos: The smart card operating system, 2024. URL `https://www.gi-de.com/en/solutions/smart-card-technology/cardos`. Información sobre el sistema operativo CardOS.

[9] Oracle. *The Java Programming Language*, 2021. URL `https://docs.oracle.com/en/java/javacard/3.1/jc_api_srvc/api_classic/index.html/`. Documen-

tación oficial sobre el paquete 'java.lang' que proporciona clases fundamentales del lenguaje Java.

[10] Oracle. *Java Card Framework API*, 2021. URL `https://docs.oracle.com/en/java/javacard/3.1/jc_api_srvc/api_classic/index.html/`. Documentación oficial sobre el paquete 'javacard.framework' que define la API del marco Java Card para el desarrollo de aplicaciones en tarjetas inteligentes.

[11] Oracle. *Java Card Security API*, 2021. URL `https://docs.oracle.com/en/java/javacard/3.1/jc_api_srvc/api_classic/index.html/`. Documentación oficial sobre el paquete 'javacard.security' que proporciona clases para la implementación de funciones de seguridad en aplicaciones Java Card.

[12] Oracle. *Java Card Runtime Environment (JCRE) Specification*, 2021. URL `https://docs.oracle.com/javacard/3.1/related-docs/JCVMS/JCVMS.pdf`. Documentación oficial sobre el entorno de ejecución de Java Card (JCRE), que define los requisitos y comportamientos del entorno de ejecución para aplicaciones Java Card.

[13] Globalplatform specifications, 2024. URL `https://globalplatform.org/`. Especificaciones oficiales de GlobalPlatform que cubren estándares y prácticas para la tecnología de tarjetas inteligentes y sistemas de gestión de confianza.

[14] Near field communication (nfc), 2024. URL `https://nfc-forum.org/`. Organización que define y promueve estándares para la tecnología de comunicación de campo cercano (NFC), que permite la comunicación entre dispositivos en proximidad.

[15] Iso/iec 18092: Information technology - telecommunications and information exchange between systems - near field communication - interface and protocol (nfc-ip1), 2014. Define el protocolo de comunicación y la interfaz para Near Field Communication (NFC), especificando las capas de comunicación y el proceso de intercambio de datos.

[16] Ecma-340: Near field communication (nfc) - interface and protocol (nfcip-1), 2008. Define la interfaz y el protocolo para NFC, especificando cómo se realiza la comunicación entre dispositivos NFC.

[17] Ecma-352: Near field communication (nfc) - interface and protocol (nfcip-2), 2011. Define el protocolo de comunicación para NFC en el modo peer-to-peer, complementando el estándar ECMA-340.

[18] Ecma-356: Near field communication (nfc) - data exchange format, 2012. Especifica el formato de intercambio de datos en NFC, incluyendo el formato de mensajes y el intercambio de datos entre dispositivos NFC.

[19] Nfc-b: Nfc forum type b tag operation, 2024. URL https://nfc-forum.org/. Especificación técnica para la operación de etiquetas NFC Tipo B según el NFC Forum.

[20] Nfc-f: Nfc forum type f tag operation, 2024. URL https://nfc-forum.org/. Especificación técnica para la operación de etiquetas NFC Tipo F según el NFC Forum.

[21] Nfc forum specifications, 2024. URL https://nfc-forum.org/. Organización que especifica los estándares para la tecnología NFC, incluyendo el formato NDEF y los Tag de Tipo 4.

[22] Nfc forum type 1 tag operation, 2024. URL https://nfc-forum.org/. Especificación técnica para la operación de etiquetas NFC Tipo 1 según el NFC Forum.

[23] Nfc forum type 2 tag operation, 2024. URL https://nfc-forum.org/. Especificación técnica para la operación de etiquetas NFC Tipo 2 según el NFC Forum.

[24] Nfc forum type 3 tag operation, 2024. URL https://nfc-forum.org/. Especificación técnica para la operación de etiquetas NFC Tipo 3 según el NFC Forum.

[25] Nfc forum tag type 4 technical specification, 2020. URL https://www.nfc-forum.org/. Especificación técnica para los Tag de Tipo 4 según NFC Forum.

[26] Iso/iec 14443: Identification cards - contactless integrated circuit cards - proximity cards, 2018. Define los requisitos para tarjetas inteligentes sin contacto en el rango de proximidad, incluyendo los métodos de comunicación y los protocolos utilizados.

[27] Iso/iec 15693: Identification cards - contactless integrated circuit cards - vicinity cards, 2006. Define los requisitos para tarjetas inteligentes sin contacto en el rango de proximidad media (vicinity), especificando los métodos de comunicación y los protocolos utilizados.

[28] Nfc forum data exchange format (ndef), 2006. Define el formato de mensajes y el intercambio de datos en tecnología NFC.

[29] Martin Paljak. Globalplatformpro, 2024. URL https://github.com/martinpaljak/GlobalPlatformPro. Repositorio de GitHub que proporciona una implementación de la API GlobalPlatform para interactuar con tarjetas inteligentes y otros dispositivos.

[30] Pc/sc scriptor, 2024. URL https://www.springcard.com/en. Herramienta para el desarrollo y pruebas de aplicaciones de tarjetas inteligentes con soporte para el protocolo PC/SC.

[31] Nfc tools, 2024. URL https://play.google.com/store/apps/details?id=com.wakdev.wdnfc&hl=es. Aplicación móvil para leer y escribir etiquetas NFC, compatible con una amplia gama de formatos y tipos de datos.

[32] Web nfc api, 2024. URL https://w3c.github.io/web-nfc/#reading-an-nfc-tag. Documentación de la API Web NFC, que proporciona especificaciones sobre la lectura y escritura de etiquetas NFC en la web.

[33] Global system for mobile communications (gsm), 2024. Estándar para redes móviles digitales, incluyendo especificaciones técnicas para la comunicación celular.